

# React.js (v18.2.0)

inky4832@daum.net

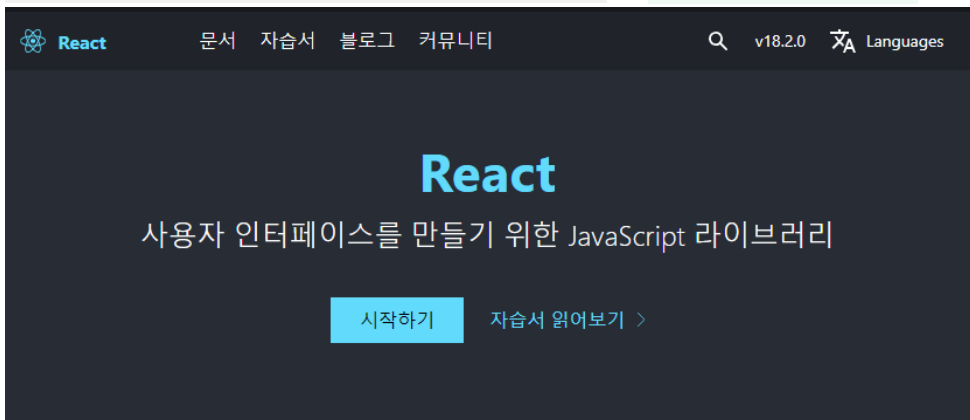
# 1장. React 개요 및 환경설정

# 1. React 홈페이지

Reactjs 18.2.0

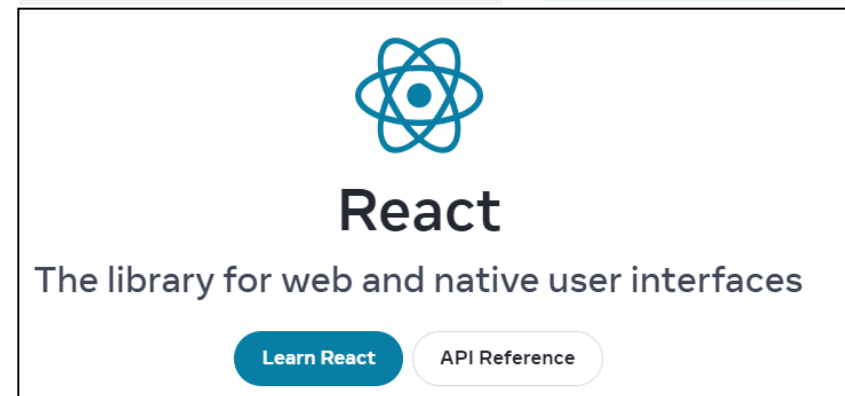
https://ko.reactjs.org

v18.2.0



https://react.dev/

V18.3.1



## 개요

facebook에서 React.js 발표.

UI 상태 자동 관리.

가상 DOM을 이용한 빠른 DOM 조작 가능.

JSX 및 자바스크립트로 정의하는 화면.

MVC의 V 담당.

Visual한 요소와 그 상태를 최신으로 유지하는데 중점을 둔 자바스크립트 라이브러리.

## 2. 다른 프레임워크와의 비교

Reactjs 18.2.0

<https://2024.stateofjs.com/ko-KR/libraries/front-end-frameworks>



### 가. node.js 설치

<http://nodejs.org>

#### 어디서든 JavaScript를 실행하세요

Node.js®는 무료, 오픈소스, 다중 플랫폼 JavaScript 런타임 환경으로 개발자 여러분이 서버, 웹 애플리케이션, 명령어 작성 도구와 스크립트를 만들도록 해줍니다.

Node.js 다운로드 (LTS) 

Node.js 다운로드 v22.13.1<sup>1</sup> LTS. Node.js는 패키지 관리자를 통해서도 다운로드 할 수 있습니다.

새로운 기능을 먼저 경험하고 싶다면 Node.js v23.7.0 <sup>1</sup>를 다운 받으세요.

```
C:\Users\Winky4>node -v  
v22.13.1
```

### 나. 개발 툴 설치 ( Visual Studio Code )

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

#### Your code editor. Redefined with AI.

Download for Windows

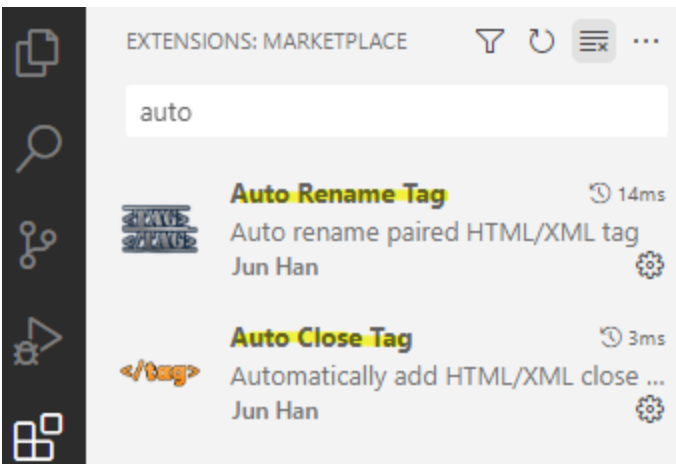
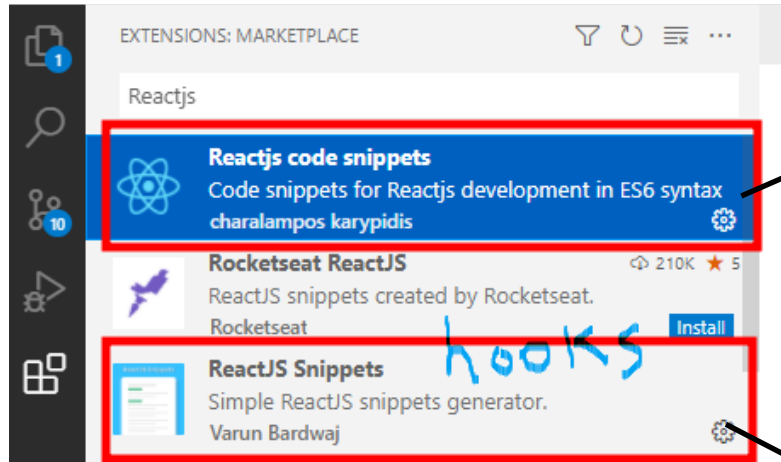
Get Copilot Free

[Web](#), [Insiders edition](#), or [other platforms](#)

## 2. React 환경 설정

Reactjs 18.2.0

### 다. VSC 확장 프로그램 설치

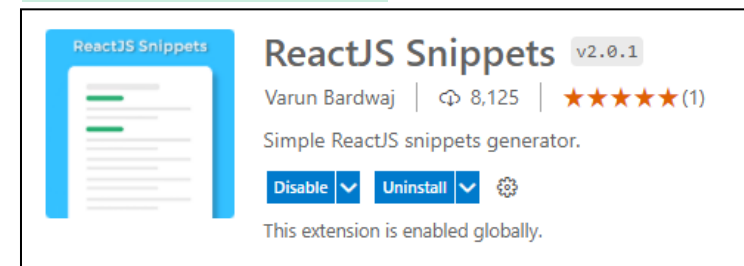


### 클래스 기반 전용



Trigger	Content
rcc→	class component skeleton
rcc→	class component skeleton with react-redux connect

### 함수 기반 전용



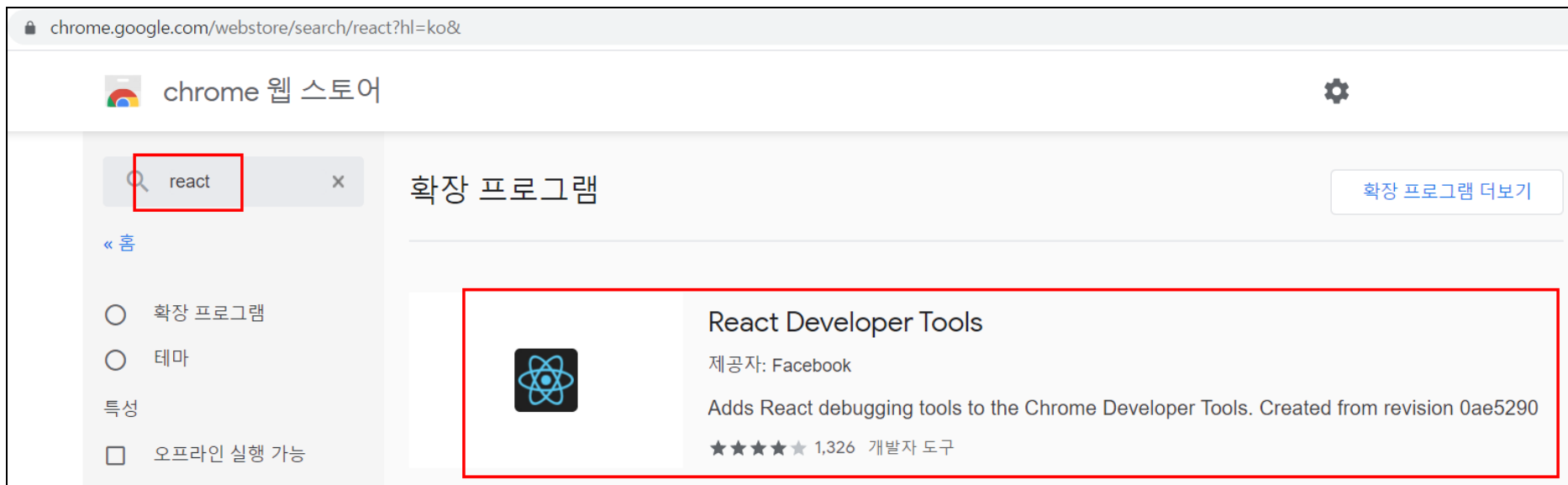
@iru	Import React/useState/useEffect/useRef
@ust	Import useState Hook
@uef	Import useEffect Hook
@urf	Import useRef Hook

## 2. React 환경 설정

Reactjs 18.2.0

라. 크롬 브라우저에 React 플러그인 설치

<https://chromewebstore.google.com/>



## 2장. my-app 프로젝트 생성



<https://ko.legacy.reactjs.org/docs/create-a-new-react-app.html#create-react-app>

## 🔗 Create React App

Create React App은 **React** 배우기에 간편한 환경입니다. 그리고 시작하기에 최고의 방법은 새로운 싱글 페이지 애플리케이션입니다.

이것은 개발 환경을 설정하고, 최신 JavaScript를 사용하게 해주며, 좋은 개발 경험과 프로덕션 앱 최적화를 해줍니다. Node 14.0.0 혹은 상위 버전 및 npm 5.6 혹은 상위 버전이 필요합니다. 새로운 프로젝트를 만들기 위해 아래의 명령어를 실행합니다.

```
npx create-react-app my-app
cd my-app
npm start
```

실행하기 전에 package.json 에서 react 버전을 18.2.0으로 수정 필요.

# 1. React toolchain 설치

Reactjs 18.2.0

```
c:\Wreactjs_study>npx create-react-app my-app
Need to install the following packages:
create-react-app@5.0.1
Ok to proceed? (y) y
```

Success! Created my-app at c:\Wreactjs\_study\my-app  
Inside that directory, you can run several commands:

```
npm start
```

Starts the development server.

```
npm run build
```

Bundles the app into static files for production.

```
npm test
```

Starts the test runner.

```
npm run eject
```

Removes this tool and copies build dependencies, configuration files and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

```
cd my-app
```

```
npm start
```

Happy hacking!

```
c:\Wreactjs_study>
```

## 2. my-app 프로젝트 실행

Reactjs 18.2.0

```
c:\Wreactjs_study>cd my-app  
c:\Wreactjs_study\my-app>npm start
```



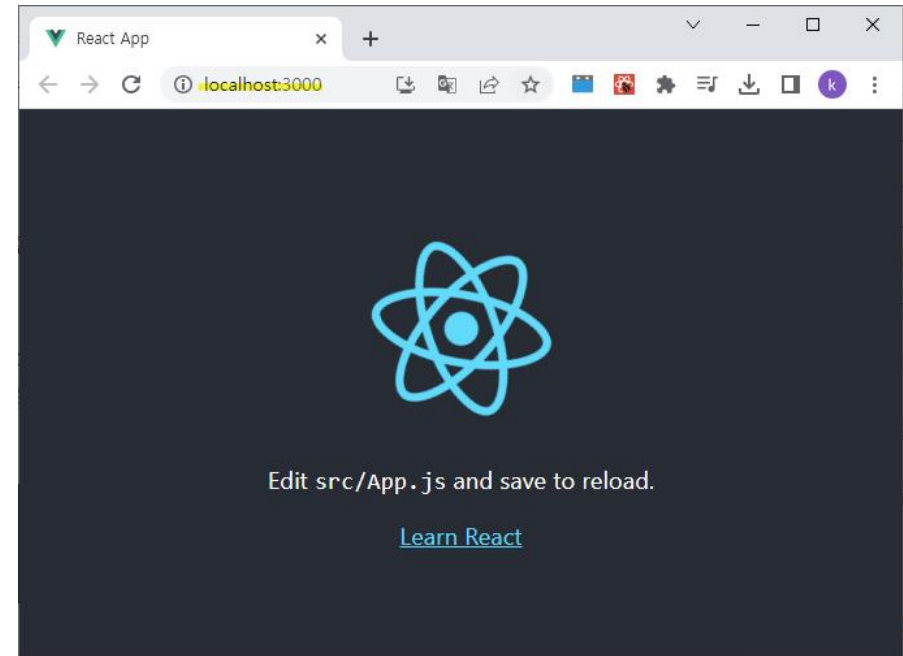
Compiled successfully!

You can now view my-app in the browser.

```
Local:      http://localhost:3000  
On Your Network: http://192.168.0.4:3000
```

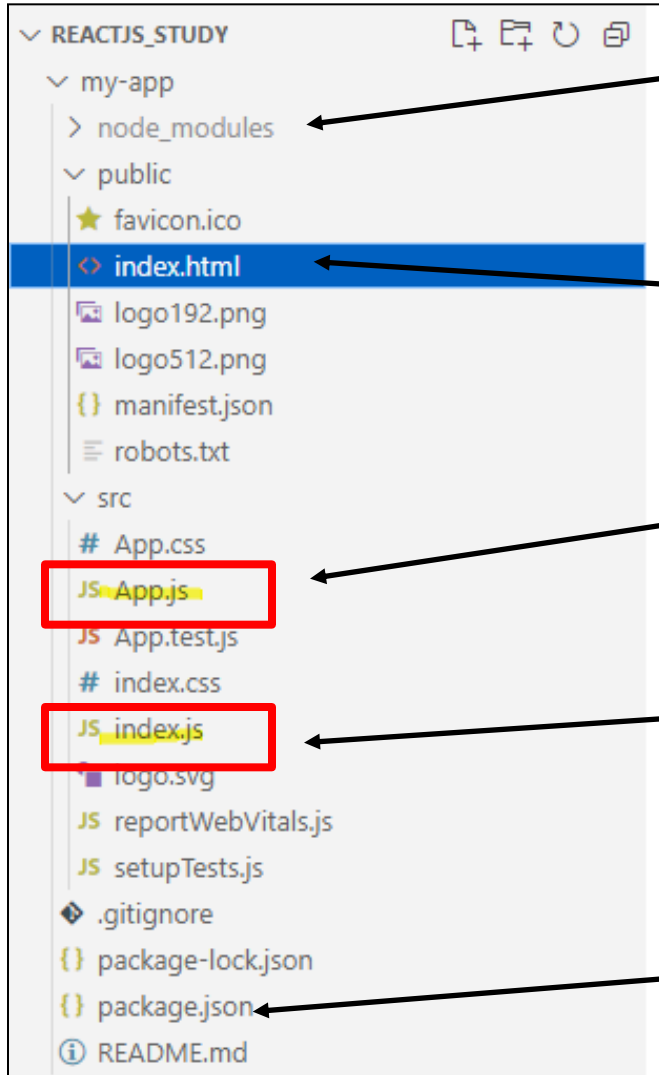
Note that the development build is not optimized.  
To create a production build, use `npm run build`.

webpack compiled successfully



### 3. my-app 프로젝트 구조

Reactjs 18.2.0



node.js는 node\_modules 폴더를 생성하고 이곳에 package.json 파일에서 명시된 외부 모듈을 다운로드 저장함.

메인 홈페이지 파일. 편집할 필요 없다.  
어플리케이션을 빌드 할 때 자동으로 모든 js 및 css 파일을 동적으로 추가된다.

React 컴포넌트 파일

Starting point. 어플리케이션의 App 컴포넌트를 로딩하여 웹브라우저에서 실행시킴.

Nodejs의 모듈 관리 설정 파일  
프로젝트에서 사용하는 third party용 패키지를 나열

## 4. App.js, index.js, index.html 관계

Reactjs 18.2.0

### App.js

```
JS App.js x
my-app > src > JS App.js > App
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
22   );
23 }
24
25 export default App;
```

### index.html

```
<> index.html x
my-app > public > <> index.html > html > body > div#root
28 </head>
29 <body>
30   <noscript>You need to enable JavaScript to run this app.</noscript>
31   <div id="root"></div>
```

### index.js

```
JS index.js x
my-app > src > JS 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
7 const root = ReactDOM.createRoot(document.getElementById('root'));
8 root.render(
9   <React.StrictMode>
10     <App />
11   </React.StrictMode>
12 );
```

## 3장. 컴포넌트 ( Component )

# 1. 컴포넌트 ( Component )

Reactjs 18.2.0

## 개요

react 앱은 컴포넌트들로 구성되고 컴포넌트는 웹 화면에서 보여지는 개별적인 화면 블록 ( 로직 + UI )을 의미한다. 컴포넌트는 버튼만큼 작을 수도 있고 전체 페이지만큼 클 수도 있으며 일반적으로 중첩된 형태로 사용된다.

<https://react.dev/learn/your-first-component#defining-a-component>

Step 1: Export the component

Step 2: Define the function

Step 3: Add markup

App.js

```
1 export default function Profile() {  
2   return (  
3       
7   )  
8 }  
9
```



# 1. 컴포넌트 ( Component )

Reactjs 18.2.0

<https://react.dev/learn#components>

## 함수형 컴포넌트 예

React 컴포넌트는 마크업을 반환하는 Javascript 함수로서 다른 컴포넌트를 포함하여 중첩 형태로 사용한다.

### MyButton 컴포넌트

```
function MyButton() {  
  return (  
    <button>I'm a button</button>  
  );  
}
```

### MyApp 컴포넌트

```
export default function MyApp() {  
  return (  
    <div>  
      <h1>Welcome to my app</h1>  
      <MyButton />  
    </div>  
  );  
}
```

**주의: 컴포넌트의 이름은 항상 대문자로 시작합니다.**

React는 소문자로 시작하는 컴포넌트를 DOM 태그로 처리합니다. 예를 들어 `<div />`는 HTML div 태그를 나타내지만, `<Welcome />`은 컴포넌트를 나타내며 범위 안에 `Welcome`이 있어야 합니다.



## 2. 함수형 컴포넌트 (Functional Component)

Reactjs 18.2.0

### 함수형 컴포넌트 형태

#### 일반 함수

```
export default function ChildFunction(props){
  return(
    <div>
      <h1>함수형 컴포넌트(Functional Component)2</h1>
      props:{props.mesg}
    </div>
  );
}
```

#### Arrow 함수

```
const ChildFunction2=(props)=>{
  return(
    <div>
      <h1>함수형 컴포넌트(Functional Component)2</h1>
      props:{props.mesg}
    </div>
  );
};

export default ChildFunction2;
```

#### destructuring Arrow 함수

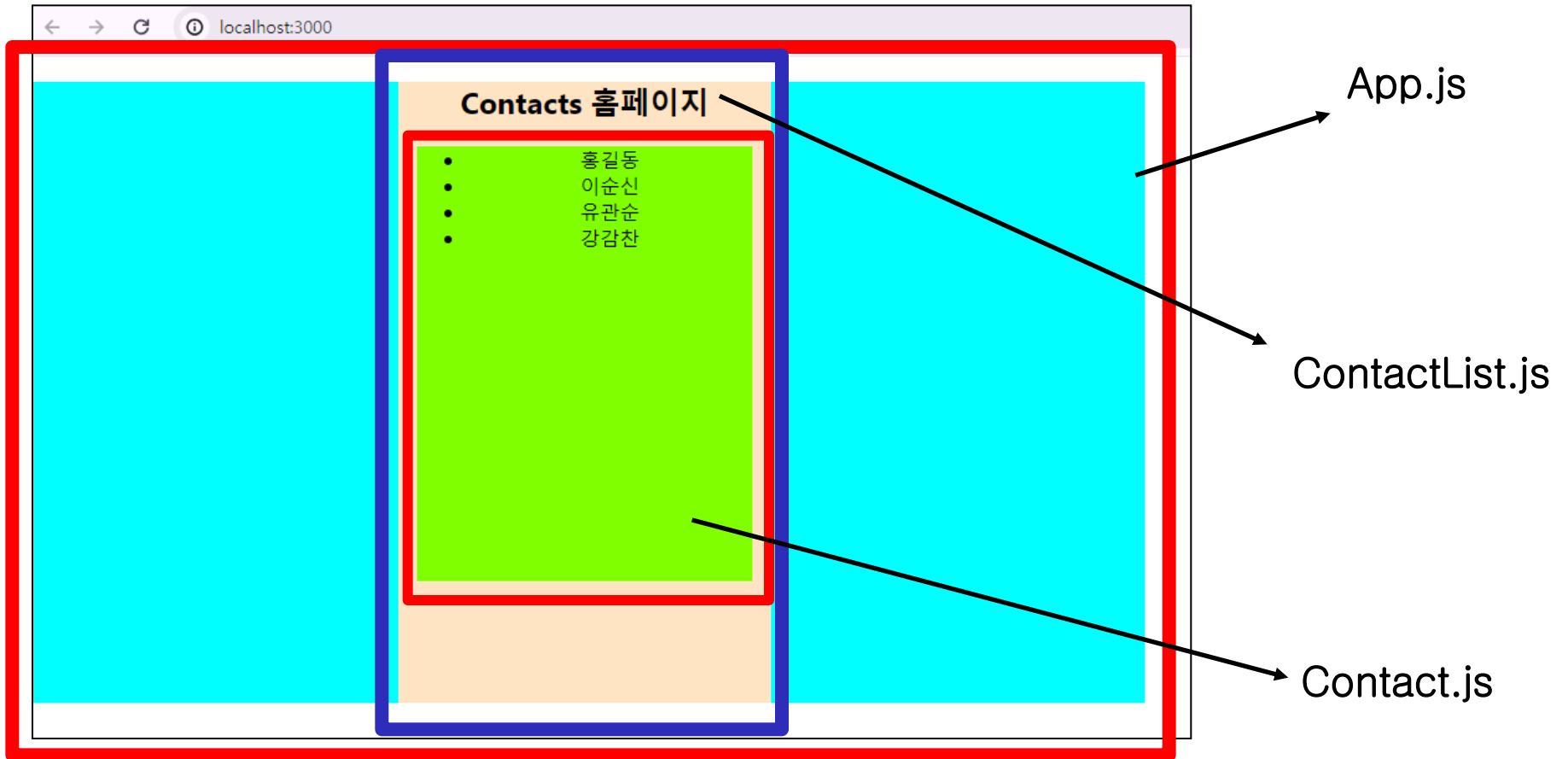
```
const ChildDestructuring=({mesg})=>{
  return(
    <div>
      <h1>함수형 컴포넌트(Functional Component)3</h1>
      props:{mesg}
    </div>
  );
};

export default ChildDestructuring;
```

### 3. 실습 1

Reactjs 18.2.0

다음과 같은 레이아웃을 가진 **함수형 컴포넌트**를 작성하시오.



## 4장. JSX

## 개요

JSX는 Javascript XML의 줄임말로 자바스크립트에 XML을 추가한 확장형 문법이다.  
JSX 문법을 사용하여 UI를 구현한다.  
JSX는 컴파일링 되면서 최적화되기 때문에 빠르다.  
컴파일 단계에서 에러를 확인할 수 있다.  
html문법과 비슷하여 더 쉽고 빠르게 UI 템플릿 작성이 가능하다.

## 사용 규칙

<https://react.dev/learn/writing-markup-with-jsx#the-rules-of-jsx>

JSX에서 모든 태그는 종료태그가 필요하다.  
JSX에서는 반드시 단 하나의 root 태그가 필요하다.  
JSX에서 변수값 출력 및 자바스크립트 코드를 사용하기 위해서는 {} 사용한다.  
JSX에서 이벤트 처리시 camel 표기법을 사용한다.  
JSX에서는 class 속성명 대신에 className 속성을 사용해야 된다.  
JSX에서 style 지정은 객체형식으로 지정하고 키값은 카멜표기법을 따른다.  
JSX에서 주석은 {/\* \*/} 을 사용해야 된다.

## 2. JSX 규칙

Reactjs 18.2.0

<https://react.dev/learn/writing-markup-with-jsx#the-rules-of-jsx>

1) JSX에서는 반드시 단 하나의 root 태그가 필수이다.

```
<div>
  <h1>Hedy Lamarr's Todos</h1>
  
  <ul>
    ...
  </ul>
</div>
```

```
<>
  <h1>Hedy Lamarr's Todos</h1>
  
  <ul>
    ...
  </ul>
</>
```

2) JSX에서 모든 태그는 종료태그가 필수이다.

```
<>
  
  <ul>
    <li>Invent new traffic lights</li>
    <li>Rehearse a movie scene</li>
    <li>Improve the spectrum technology</li>
  </ul>
</>
```

## 2. JSX 규칙

Reactjs 18.2.0

3) JSX에서는 class 대신에 className 속성을 사용한다.  
(class 키워드는 JS의 클래스와 키워드가 중복됨)

```

```

4) JSX에서 이벤트 처리시 camel 표기법 필수이다.

```
function App() {  
  return (  
    <div>  
      <Menu /><br/>  
      <button onClick={()=>console.log("OK")}>OK</button>  
    </div>  
  );  
}
```

## 2. JSX 규칙

Reactjs 18.2.0

5) JSX에서 style은 객체 형식으로 사용하고 속성명은 camel 표기법 사용.

```
function App() {  
  return (  
    <div>  
      <Menu /><br/>  
      <button onClick={()=>console.log("OK")}>OK</button>  
      <p style={{fontSize:'20px', backgroundColor:'red'}}>Hello</p>  
    </div>  
  );  
}
```

6) JSX에서 변수값 출력 및 자바스크립트 코드 작성할 때 {} 사용한다.

<https://react.dev/learn/javascript-in-jsx-with-curly-braces>

```
let title = "Menu"  
function Menu(props) {  
  return (  
    <div>  
      {title} 입니다.  
    </div>  
  );  
}
```

```
let title = "Menu"  
let names = ["홍길동", "이순신", "강감찬"]  
function Menu(props) {  
  return (  
    <div>  
      {title} 입니다.<br />  
      <ul>  
        {  
          names.map((row, idx)=>{  
            return <li key={idx}>{row}</li>  
          })  
        }  
      </ul>  
    </div>  
  );  
}
```

### 7) JSX에서 spread 연산자 사용 가능하다.

```
let attr = { href: "http://www.google.com", target: "_blank" }  
function Menu(props) {  
  return (  
    <div>  
      {title} 입니다.<br />  
      <a {...attr}>구글</a>  
    </div>  
  )  
}
```

### 8) JSX에서 주석은 { /\* \*/ } 형식이다.

```
return (  
  <div>  
    { /* 이것은 주석입니다 */ }  
    <p style={{ fontSize: '20px', background: 'red' }}>world</p>  
    <p style={myStyle}>happy</p>  
  </div>  
) ;
```



## 5장. Props 속성

<https://react.dev/learn/passing-props-to-a-component>

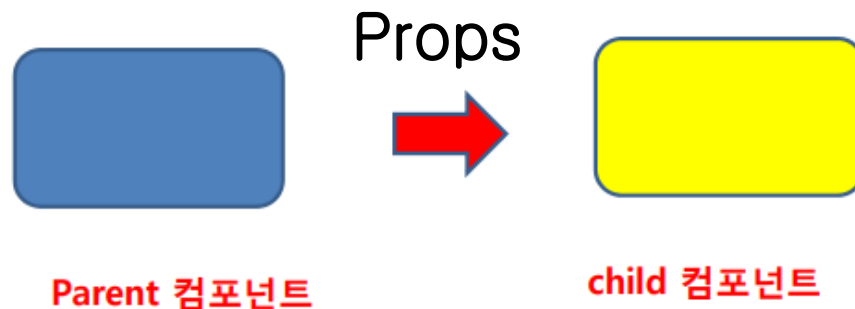
## 개요

컴포넌트에서 사용할 데이터 중에서 변경되지 않는 (immutable) 데이터를 처리할 때 사용한다.

일반적으로 부모(parent) 컴포넌트에서 자식(child) 컴포넌트로 데이터를 전달할 때 props를 사용한다. (읽기 전용)

React에서 모든 데이터의 흐름은 단방향으로 처리한다.

However, props are **immutable**—a term from computer science meaning “unchangeable”. When a component needs to change its props (for example, in response to a user interaction or new data), it will have to “ask” its parent component to pass it *different props*—a new object! Its old props will then be cast aside, and eventually the JavaScript engine will reclaim the memory taken by them.



## 2. Props 사용

Reactjs 18.2.0

### 원본

#### App.js

```
1 function Avatar() {
2   return (
3     
10  );
11 }
12
13 export default function Profile() {
14   return (
15     <Avatar />
16   );
17 }
```



### Step 1: Pass props to the child component

```
export default function Profile() {
  return (
    <Avatar
      propsName="value"
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}
      size={100}
    />
  );
}
```

### Step 2: Read props inside the child component

```
function Avatar({ person, size }) {
  return (
    <img
      className="avatar"
      src={getImageUrl(person)}
      alt={person.name}
      width={size}
      height={size}
    />
  );
}
```

```
function Avatar(props) {
  let person = props.person;
  let size = props.size;
}
```

자식에 JSON형식으로  
병합되어 전달된다.

```
export function getImageUrl(person, size = 's') {
  return (
    'https://i.imgur.com/' +
    person.imageId +
    size +
    '.jpg'
  );
}
```

### 3. default Props 값 설정

Reactjs 18.2.0

<https://ko.reactjs.org/docs/typechecking-with-proptypes.html#default-prop-values>

- 기본적으로 Props 속성은 필수가 아니다.
- 기본값을 설정하기 위해서는 컴포넌트 함수(클래스) 하단에

`className.defaultProps = { propName: value }` 를 삽입;

```
function App() {  
  return (  
    <div>  
      <ChildComponent name="홍길동" age={20} />  
      <hr />  
      <ChildComponent />  
    </div>  
  );  
}
```

```
function ChildComponent({ name, age }){  
  // Props 얻기  
  // const { name, age } = props;  
  return (  
    <div>  
      이름:{name}<br />  
      나이:{age}  
    </div>  
  );  
}
```

```
//기본 Props 설정 : rdp  
ChildComponent.defaultProps = {  
  name: "유관순",  
  age: 18  
};
```

```
export default ChildComponent;
```

← → ↻ ⓘ localhost:3000
이름:홍길동 나이:20
이름:유관순 나이:18

## 4. 실습 2

Reactjs 18.2.0

### App.js

```
import PersonList from "../components/PersonList";

let persons = [
  { name: "홍길동", age: 20 },
  { name: "이순신", age: 30 },
  { name: "유관순", age: 40 },
  { name: "강감찬", age: 50 },
];

function App() {
  return (
    <>
      <h1>학생 정보</h1>
    </>
  );
}
```

### 학생 목록 실행 결과

React App

localhost:3000

### 학생 정보

번호	이름	나이
1	홍길동	20
2	이순신	30
3	유관순	40
4	강감찬	50

App.js

PersonList.js

## 5. 실습 3

Reactjs 18.2.0

### 학생 목록 실행 결과

학생 정보

번호	이름	나이
1	홍길동	20
2	이순신	30
3	유관순	40
4	강감찬	50

App.js

Person.js  
(table 태그의 <tr> 구현)

PersonList.js

## 6. JSX 전달

Reactjs 18.2.0

<https://react.dev/learn/passing-props-to-a-component#passing-jsx-as-children>

```
1 import Avatar from './Avatar.js';
2
3 function Card({ children }) {
4   return (
5     <div className="card">
6       {children}
7     </div>
8   );
9 }
10
11 export default function Profile() {
12   return (
13     <Card>
14       <Avatar
15         size={100}
16         person={{
17           name: 'Katsuko Saruhashi',
18           imageId: 'Yfe0qp2'
19         }}
20       />
21     </Card>
22   );
23 }
24
```

props.children 으로 받음

body 로 전달

## 7. Forwarding props 패턴

Reactjs 18.2.0

Props를 여러 단계의 하위 컴포넌트로 전달하는 효율적인 방법으로 merge와 destructuring 방법을 활용.

App 컴포넌트



First 컴포넌트



Second 컴포넌트

```
function App() {  
  return (  
    <div>  
      <FirstChild name="홍길동" age={20} />  
    </div>  
  );  
}
```

```
function FirstChild(props){  
  let user = {  
    ...props,  
    address: "서울"  
  }  
  return (  
    <div>  
      <SecondChild {...user} />  
    </div>  
  );  
}
```

merge

destructuring

```
function SecondChild(props){  
  // Props 얻기  
  const { name, age, address } = props;  
  return (  
    <div>  
      이름:{name}<br />  
      나이:{age}<br />  
      주소:{address}<br />  
    </div>  
  );  
}
```



## 6장. 이벤트 처리

<https://ko.reactjs.org/docs/handling-events.html>

## 개요

React에서 이벤트를 처리하는 방식은 DOM 엘리먼트에서 이벤트를 처리하는 방식과 매우 유사하다.

몇 가지 문법 차이는 다음과 같다.

React의 이벤트는 소문자 대신에 camel case를 사용한다.

JSX를 사용하여 문자열이 아닌 {함수} 형식으로 이벤트 핸들러를 전달한다.

false값 대신에 반드시 preventDefault를 명시적으로 호출해야 된다.

## 기존 HTML의 이벤트 처리

```
<button onclick="activateLasers()">  
  Activate Lasers  
</button>
```

## React의 이벤트 처리

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```

## 2. 기본 동작 및 이벤트전파 방지

Reactjs 18.2.0

### 기존 HTML의 기본동작 방지

```
<form onsubmit="console.log('You clicked submit.');" return false">  
  <button type="submit">Submit</button>  
</form>
```

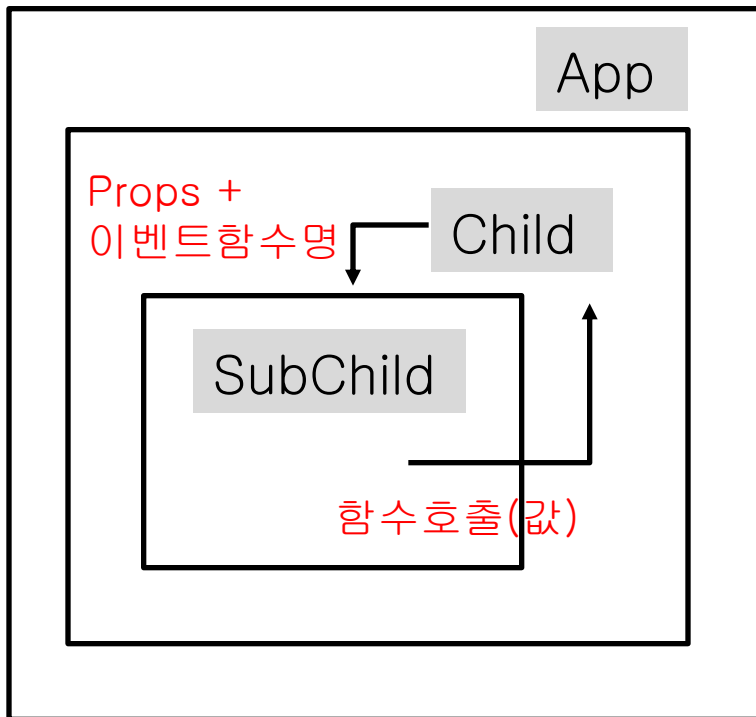
### React의 기본동작 방지

```
function Form() {  
  function handleSubmit(e) {  
    e.preventDefault();  
    console.log('You clicked submit.');  }  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <button type="submit">Submit</button>  
    </form>  
  );  
}
```

### 3. 계층구조 Events 처리

Reactjs 18.2.0

부모의 이벤트 함수를 자식에서 호출할 수 있는 방법으로서 Props를 활용한다.  
이 방법을 활용하면 자식에서 부모로 데이터를 전달할 수 있다.



```
function ChildComponent(props) {  
  // 콜백함수  
  function handleChildEvent() {  
    console.log("handleChildEvent");  
  }  
  
  return (  
    <div>  
      <SubChildComponent onEvent={handleChildEvent}/>  
    </div>  
  );  
}
```

```
function SubChildComponent(props) {  
  const { onEvent } = props;  
  return (  
    <div>  
      <h2>ChildComponent 콜백 호출</h2>  
      <button onClick={onEvent}>call1</button>  
    </div>  
  );  
}
```

## 7장. hooks 개념 및 상태관리

<https://ko.legacy.reactjs.org/docs/hooks-intro.html>

<https://ko.legacy.reactjs.org/docs/hooks-faq.html#gatsby-focus-wrapper>

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

## 개요

함수형 컴포넌트에서 클래스 컴포넌트의 기능을 사용할 수 있도록 해주는 기능.

React 16.8 버전(2019년)에 추가된 공식 라이브러리.

클래스 컴포넌트에서만 사용했던 state와 라이프사이클을 함수형 컴포넌트에서도 사용 가능.

공식 문서에서는 클래스 컴포넌트보다 함수형 컴포넌트 사용 권장.

## 사용 규칙

가. 최상위 함수내에서만 hook을 호출한다. (반복문, 조건문, 중첩된 함수등에서 호출 안됨)

나. React 함수에서만 hook을 호출한다. (일반적인 JS 함수에서는 호출 안됨)

다. 커스텀 hook 작성시 use 접두어 사용한다.

라. React는 hook 호출되는 순서에 의존한다. (여러 개 사용되는 경우 순차적으로 동작됨)

## 종류

useState hook: 동적 상태 관리

useEffect hook: 부수 효과 관리(side effect)

useMemo hook: 연산 값 재사용

useCallback hook: 특정 함수 재사용

useRef hook: DOM 참조

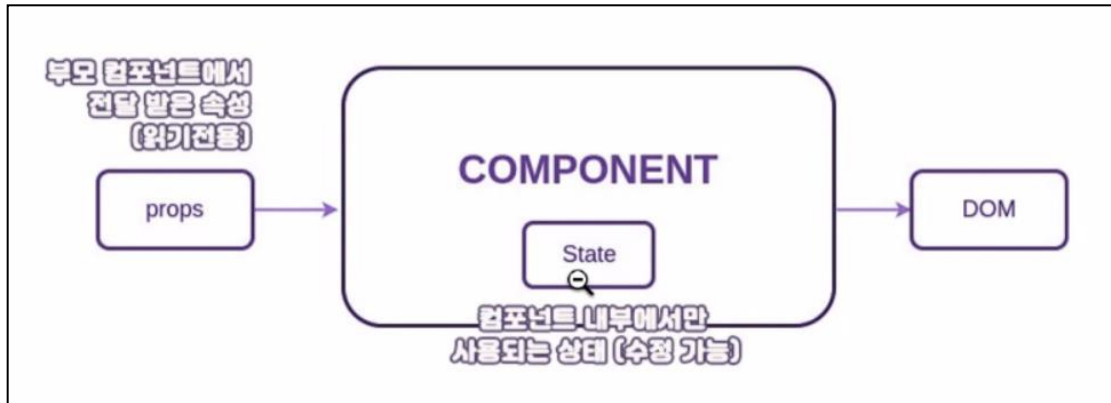
useReducer hook: useState hook의 업그레이드 버전 ( 컴포넌트와 state 관리 로직 분리)

useContext hook: 전역 데이터 관리

## 2. 상태관리

Reactjs 18.2.0

<https://ko.reactjs.org/docs/state-and-lifecycle.html>



```
1: import React, { useState } from 'react';
2:
3: function Example() {
4:   const [count, setCount] = useState(0);
5:
6:   return (
7:     <div>
8:       <p>You clicked {count} times</p>
9:       <button onClick={() => setCount(count + 1)}>
10:        Click me
11:      </button>
12:    </div>
13:  );
14: }
```



### 3. useState hook

Reactjs 18.2.0

<https://ko.reactjs.org/docs/hooks-state.html>

<https://react.dev/learn/state-a-components-memory>

#### state 개요

컴포넌트에서 사용되는 **변경 가능한 데이터(mutable)**를 다룰 때 사용한다.  
state 값이 변경되면 자동으로 화면이 재 랜더링 된다.

#### 주요 특징

state에 저장되는 객체(배열)는 반드시 불변객체로 관리해야 된다.  
(배열 요소 값을 수정하는 방식이 아닌 배열 자체를 덮어쓰는 방식)  
상태값 변경함수는 비동기이면서 배치로 실행되고 값 설정시 arrow 함수로 사용  
이 가능하다.  
가상 DOM 과 실제 DOM을 비교하는 방식으로 처리되기 때문에 매우 효율적으로  
화면 처리가 된다.

#### state 사용 방법

가. import

```
import {useState} from 'react'
```

나. 초기화는 함수안에서

```
const [변수, 변경함수] = useState(변수초기값)
```

다. JSX에서 값 출력은 {변수} 형식을 사용

라. 값 수정은 변경함수(변경값|arrow 함수) 형식을 사용.

값 변경 후 자동으로 화면이 재 랜더링 됨.

## 4. useState 이용한 조건부 렌더링

Reactjs 18.2.0

<https://react.dev/learn/conditional-rendering>

### 1. If 문 이용

```
if (isPacked) {  
  return <li className="item">{name} ✓</li>;  
}  
return <li className="item">{name}</li>;
```

### 3. && 이용

```
return (  
  <li className="item">  
    {name} {isPacked && '✓'}  
  </li>  
);
```

### 2. 3항 연산자 이용

```
return (  
  <li className="item">  
    {isPacked ? name + ' ✓' : name}  
  </li>  
);
```

<https://react.dev/learn/conditional-rendering>

### 1. JSON 형태

```
const [person, setPerson] = useState({
  firstName: 'Barbara',
  lastName: 'Hepworth',
  email: 'bhepworth@sculpture.com'
});
```

```
setPerson({
  ...person, // Copy the old fields
  firstName: e.target.value // But override this one
});
```

## 5. useState 이용한 불변 객체 이슈

Reactjs 18.2.0

<https://react.dev/learn/updating-arrays-in-state>

### 2. 배열 형태

	avoid (mutates the array)	prefer (returns a new array)
adding	<code>push</code> , <code>unshift</code>	<code>concat</code> , <code>[...arr]</code> spread syntax ( <a href="#">example</a> )
removing	<code>pop</code> , <code>shift</code> , <code>splice</code>	<code>filter</code> , <code>slice</code> ( <a href="#">example</a> )
replacing	<code>splice</code> , <code>arr[i] = ...</code> assignment	<code>map</code> ( <a href="#">example</a> )
sorting	<code>reverse</code> , <code>sort</code>	copy the array first ( <a href="#">example</a> )

## 5. useState 이용한 불변 객체 이슈

Reactjs 18.2.0

```
const [artists, setArtists] = useState([]);
```

### 재 랜더링 안됨

```
<button onClick={() => {  
  artists.push({  
    id: nextId++,  
    name: name,  
  });  
}}>Add</button>
```

### 재 랜더링 됨

```
<button onClick={() => {  
  setArtists([  
    ...artists,  
    { id: nextId++, name: name }  
  ]);  
}}>Add</button>
```

## 6. Two way binding ( useState + input )

Reactjs 18.2.0

<https://react.dev/learn/reacting-to-input-with-state#step-5-connect-the-event-handlers-to-set-state>

### 개념

input 태그의 value 속성에 state를 사용할 때는 onChange 이벤트 처리가 필수. 키보드 입력시 state에 변경값 적용을 onChange 에서 처리한다.

```
const [answer, setAnswer] = useState('');
```

```
<form onSubmit={handleSubmit}>
  <textarea
    value={answer}
    onChange={handleTextareaChange}
    disabled={status === 'submitting'}
  />
```

```
function handleSubmit(e) {
  e.preventDefault();
  setStatus('submitting');
```

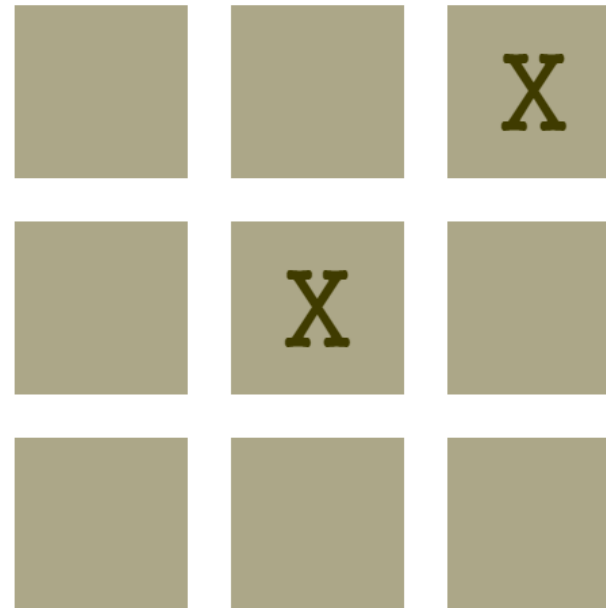
```
function handleTextareaChange(e) {
  setAnswer(e.target.value);
}
```

### 함수형 컴포넌트 구현

#### 1. 초기화면



#### 2. 선택시





## 8. 실습 5

Reactjs 18.2.0

### 함수형 컴포넌트 구현

#### 1. 초기화면

← → ↻ ⓘ localhost:3000

**state 실습**

step: 1 ▾

num: 0

- +

#### 2. Step 선택

← → ↻ ⓘ localhost:3000

**state 실습**

step: 1 ▾

num: 1

- +

#### 3. + 선택

← → ↻ ⓘ localhost:3000

**state 실습**

step: 2 ▾

num: 6

- +

#### 4. - 선택

← → ↻ ⓘ localhost:3000

**state 실습**

step: 2 ▾

num: 1

- +

음수값 허용 불가,  
0으로 초기화

## 함수형 컴포넌트 구현

### 1. 초기화면

localhost:3000

이름

나이

주소

저장

### 2. 입력 화면

localhost:3000

이름

나이

주소

저장

### 3. 저장 및 출력

localhost:3000

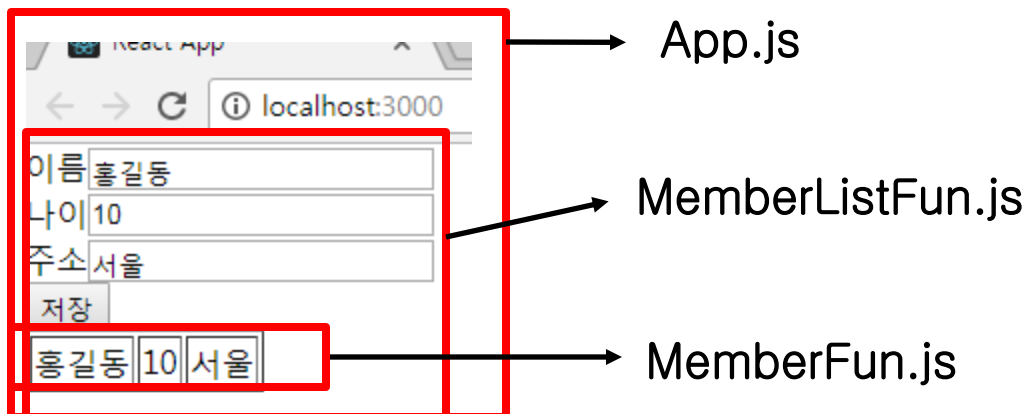
이름

나이

주소

저장

홍길동 20 서울



```
function MemberListFun(props) {

  const [memberData, setMemberData] = useState([])

  const [inputs, setInputs] = useState({
    username: '',
    age: '',
    address: ''
  });
}
```

## 8장. DOM 참조, Context 및 부수 효과 ( side effects)

<https://ko.legacy.reactjs.org/docs/hooks-reference.html#useref>

<https://react.dev/reference/react/useRef#referencing-a-value-with-a-ref>

## 개요

일반적으로 생성된 DOM 노드 또는 자식 컴포넌트에 직접 접근할 때 주로 사용되고 current 속성으로 참조한다.

state와 다르게 값을 변경해도 재 렌더링 되지 않기 때문에 화면에 보여주기 위한 정보를 저장하는 용도로는 적합하지 않다.

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onClick}>Focus the input</button>
    </>
  );
}
```

## 문법1: 일반값

```
import { useRef } from 'react';

export default function Counter() {
  let ref = useRef(0);

  function handleClick() {
    ref.current = ref.current + 1;
    alert('You clicked ' + ref.current + ' times!');
  }

  return (
    <button onClick={handleClick}>
      Click me!
    </button>
  );
}
```

## 문법2: DOM 참조

```
import { useRef } from 'react';

function MyComponent() {
  const inputRef = useRef(null);
  // ...

  // ...
  return <input ref={inputRef} />;

  function handleClick() {
    inputRef.current.focus();
  }
}
```

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

## forwardRef(자식)

부모 컴포넌트에서 자식 컴포넌트 접근시 사용되며 forwardRef(자식) 를 사용하면 자식 컴포넌트를 부모에게 노출시킬 수 있게 된다.

자식 컴포넌트에서는 부모에서 전달된 ref는 props로 처리가 안되기 때문에 props와 별개로 ref 파라미터를 설정해야 된다.

## 부모 컴포넌트 예

```
return (  
  <form>  
    <MyInput label="Enter your name:" ref={ref} />  
    <button type="button" onClick={handleClick}>  
      Edit  
    </button>  
  </form>  
)
```

## 자식 컴포넌트 예

```
import { forwardRef } from 'react';  
  
const MyInput = forwardRef(function MyInput(props, ref) {  
  // ...  
});
```

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

## useImperativeHandle()

forwardRef 함수내에서 부모에서 호출할 자식함수는 useImperativeHandle()로 감싸야 부모에게 메서드를 노출시킬 수 있다.

`useImperativeHandle(ref, createHandle, dependencies?)` 

Call `useImperativeHandle` at the top level of your component to customize the ref handle it exposes:

```
import { forwardRef, useImperativeHandle } from 'react';

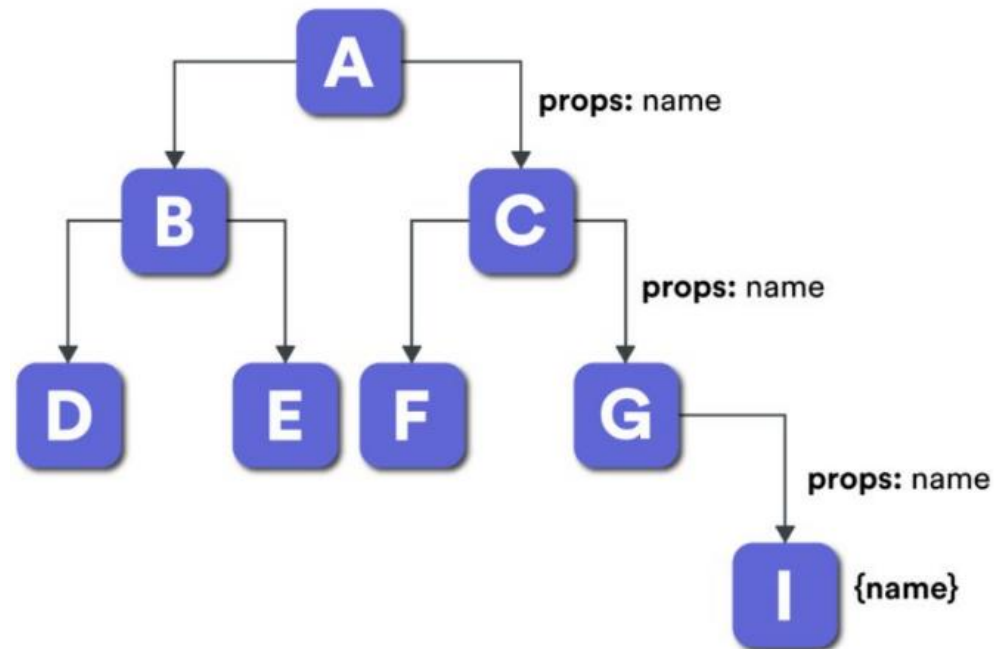
const MyInput = forwardRef(function MyInput(props, ref) {
  useImperativeHandle(ref, () => {
    return {
      // ... your methods ...
    };
  }, []);
  // ...
```

## 2. useContext/createContext hook

Reactjs 18.2.0

### 개요

대부분의 어플리케이션은 다음과 같이 구성될 확률이 매우 높다.  
이러한 상황을 Prop Drilling 이라고 부른다.  
이러한 경우 쓸데없는 코드가 추가되고 재사용이 힘들게 된다.





## 2. useContext/createContext hook

Reactjs 18.2.0

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

### 구현

```
# 부모코드
const UserContext = createContext('기본값');
<UserContext.Provider value={name}>
  <Child />
</ UserContext.Provider >
# 자식코드
const username = useContext(UserContext);
<p>`${username}님 안녕하세요`</p>
```

### 특징

값을 전달 받을 수 있는 컴포넌트의 수에 제한은 없음.  
Provider 하위에 또 다른 Provider 를 사용하는 중첩 형태 가능하고 이 경우 하위 Provider의 값이 우선 적용됨.  
Provider 하위에서 context 를 구독하는 모든 자식 컴포넌트는 Provider의 value prop이 바뀔 때마다 자동으로 다시 렌더링 된다.

## 2. useContext/createContext hook

Reactjs 18.2.0

### 최상위 컴포넌트 코드

```
import { createContext, useContext } from 'react';

const ThemeContext = createContext(null);

export default function MyApp() {
  return (
    <ThemeContext.Provider value="dark">
      <Form />
    </ThemeContext.Provider>
  )
}
```

### 중간 컴포넌트 코드

```
function Form() {
  return (
    <Panel title="Welcome">
      <Button>Sign up</Button>
      <Button>Log in</Button>
    </Panel>
  );
}
```

### 최하위 컴포넌트 코드

```
import { useContext } from 'react';

function Button() {
  const theme = useContext(ThemeContext);
  // ...
}
```

## 2. useContext/createContext hook

Reactjs 18.2.0

### 값과 함수를 동시에 전달하는 코드 예

```
function MyApp() {  
  const [currentUser, setCurrentUser] = useState(null);  
  
  function login(response) {  
    storeCredentials(response.credentials);  
    setCurrentUser(response.user);  
  }  
  
  return (  
    <AuthContext.Provider value={{ currentUser, login }}>  
      <Page />  
    </AuthContext.Provider>  
  );  
}
```

### 3. useEffect hook

Reactjs 18.2.0

<https://ko.reactjs.org/docs/hooks-effect.html>

<https://react.dev/learn/lifecycle-of-reactive-effects#the-lifecycle-of-an-effect>

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

#### 주요 기능

현재 랜더링 사이클에 영향을 주지 않는 기능으로서 이 상황이 반응앱의 컨텍스트 입장에서는 부작용(side effect)이라고 할 수 있다.

핵심 기능은 생성/수정 기능인 초기화 작업과 제거기능(cleanup)인 자원반납 코드를 하나의 useEffect 함수에서 모두 구현할 수 있다.

예> 서버 API 연동 및 DOM 접근, 이벤트추가/삭제 등

#### 문법

**useEffect( 익명함수, [의존성배열])**

익명함수는 비동기로 동작되고 초기화 및 cleanup 기능을 구현한다.

#### 특징

`useEffect( 익명함수, [의존성배열])`

`useState`, `useRef`와 다르게 반환값이 없다.

익명함수는 바로 실행되지 않고 App이 실행되어 DOM이 랜더링 된 후에 비동기로 실행된다.

만약 state값이 변경되면 App이 다시 실행되기 때문에 이론적으로 `useEffect` 함수도 다시 실행된다.

이때 이것을 다시 실행할지 안 할지 여부를 [의존성배열]로 조절할 수 있다.

\* 의존성 배열 동작 방식 정리

[ ] 처럼 비어있는 배열을 지정하면 종속값이 없기 때문에 `useEffect`는 단 한번만 실행됨.

[변수] 배열을 지정하면 변수값이 변경될 때마다 `useEffect`는 재실행됨.

배열 자체를 지정하지 않으면 App이 다시 실행될 때 `useEffect`도 재실행됨.

#### cleanup 기능 구현

`useEffect( 익명함수, [의존성배열])`

익명함수에서 cleanup 기능의 함수를 return 해서 구현한다.

cleanup 함수는 맨 처음 실행될 때는 수행 안되고 다음 부수효과함수가 호출되기 직전 및 컴포넌트가 사라지기 직전(unmount)에 호출된다.

따라서 [] 빈 배열을 지정하면 재실행이 안되기 때문에 cleanup 기능을 구현할 수 없다.

예>

```
useEffect(()=>{
  console.log("number 값이 변경됨.")
  //clean up 기능
  return ()=>{console.log("clean up.....")}
},[number])

..
console.log("App 호출");
```

#### 대표적인 useEffect 사용 예

- 1) DOM 컨트롤 ( 실제 문서 접근 및 조작 )
- 2) 네트워크 통신 ( 비동기 통신 요청 및 응답 )
  - fetch API 또는 axios 라이브러리
- 3) 이벤트 핸들링 작업에서의 구독 및 취소 작업
  - 컴포넌트 생성시점에 이벤트를 구독하고 제거시점에 이벤트를 취소.

## 9장. 폼 처리 및 http 요청



## React 의 폼 처리 특징

state 이용한 경우에는 기본적으로 작성한 폼 태그에 입력이 안됨.

React에서 폼 처리를 하기 위해서는 반드시 다음 2가지를 고려해야 한다.

- 1) React의 state 값
- 2) DOM 자체의 value 값 ( 키보드로 입력하면 화면이 수정됨 )

위의 2개값 모두 값이 변경되면 화면이 수정되기 때문에 react 에서 폼 양식 요소를 사용할 때는 이 2개값이 반드시 동기화 되어야 한다.

```
const [username, setUsername] = useState('');

function handleUsernameChange(event){
  setUsername(event.target.value);
}

<input type="text" name="username" value={username}
      onChange={handleUsernameChange}
/>
```

### 1.state + onChange 이용

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

const [inputs, setInputs] = useState({ email: '', password: '' });

function handleInputChange(identifier, value) {
  setInputs((prevValues) => ({
    ...prevValues,
    [identifier]: value,
  }));
}

<input name="email"
  onChange={(event) => handleInputChange('email', event.target.value)}
  value={inputs.email}
/>
```

### 2. ref 이용

```
import { useRef } from 'react';

const email = useRef();
const password = useRef();

function handleSubmit(event) {
  event.preventDefault();
  const enteredEmail = email.current.value;
  const enteredPassword = password.current.value;
  console.log(enteredEmail, enteredPassword);
}

<input name="email"
      ref={email}
/>
```

### 3. FormData 객체 이용

```
export default function Signup() {  
  function handleSubmit(event) {  
    event.preventDefault();  
  
    const fd = new FormData(event.target);  
    const email = fd.get("email");  
    const password = fd.get("password");  
  }  
  <input name="password" />  
  <input name="email" />  
}
```

#### Blur 시점에 처리하기

```
/// 유효성 체크용
const [didEdit, setDidEdit] = useState({email: false, password: false});

//유효성 조건 지정
const emailsInvalid = didEdit.email && !inputs.email.includes('@');
const passwordsInvalid = didEdit.password && !(inputs.password.length > 6);

function handleInputChange(identifier, value) {
  ...
  /// 유효성 체크용
  setDidEdit((prevEdit) => ({
    ...prevEdit,
    [identifier]: false,
  }));
}
```

<https://ko.reactjs.org/docs/faq-ajax.html>

자바스크립트 환경에서 사용 가능한 비동기 Ajax 통신 방법은 다음과 같다.

- 1) XMLHttpRequest 및 jQuery Ajax
- 2) window.fetch 함수
- 3) axios 라이브러리

### window.fetch 함수

```
function App() {  
  const [usersList, setUsersList] = useState([]);  
  useEffect(() => {  
    const xxx = async function () {  
      const response = await fetch('https://reqres.in/api/users?page=2');  
      const resData = await response.json();  
      const usersList = resData.data;  
      setUsersList(usersList);  
    }  
    xxx();  
  }, []);  
}
```

<https://github.com/axios/axios>

### Installing

Using npm:

```
$ npm install axios
```

### axios API

Requests can be made by passing the relevant config to `axios`.

`axios(config)`

```
// Send a POST request
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

Performing a `POST` request

```
axios.post('/user', {
  firstName: 'Fred',
  lastName: 'Flintstone'
})
.then(function (response) {
  console.log(response);
})
.catch(function (error) {
  console.log(error);
});
```

`axios.request(config)`

`axios.get(url[, config])`

`axios.delete(url[, config])`

`axios.head(url[, config])`

`axios.options(url[, config])`

`axios.post(url[, data[, config]])`

`axios.put(url[, data[, config]])`

`axios.patch(url[, data[, config]])`

## 6. axios config 객체

Reactjs 18.2.0

axios 함수에 전달하는 config 객체는 다음과 같은 주요 속성을 갖는다.

```
let axios = axios({
  url: "./food.json", // 호출할 서버의 경로

  method: "get", // 사용하는 http method(post, get, put, delete)로 default는 get

  params: {
    name: "hong"
  }, // url 즉 쿼리스트링을 구성하는 파라미터 요소

  data: {
    age: 10,
    addr: "seoul"
  }, // request body를 통해서 서버로 전송되는 값(post, put, patch에서 사용)
});
```

요청에 대한 응답 결과는 then과 catch 콜백함수로 처리한다.

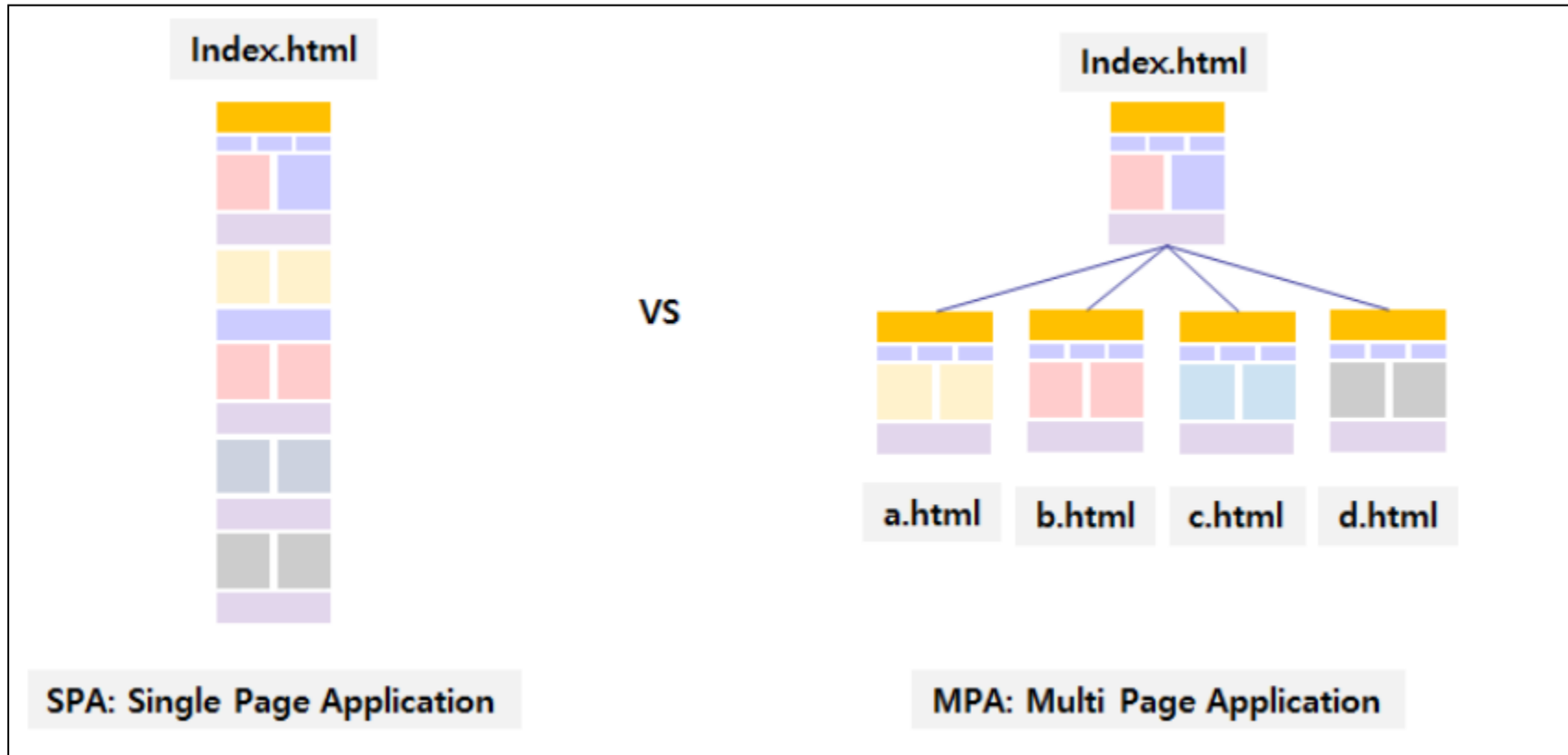
```
axios.then(
  success_callback
).catch(
  error_callback
).finally(
  finally_callback
);
```

```
{
  // 서버가 출력한 값은 언제나 data 속성 안에 존재한다.
  data: {},
  // HTTP status code
  status: 200,
  // HTTP status message from the server response
  statusText: 'OK',
  // `headers` the headers that the server responded with All header names are lower cased
  headers: {},
  // `config` is the config that was provided to `axios` for the request
  config: {},
}
```



## 10장. 라우팅 v 6.23.1

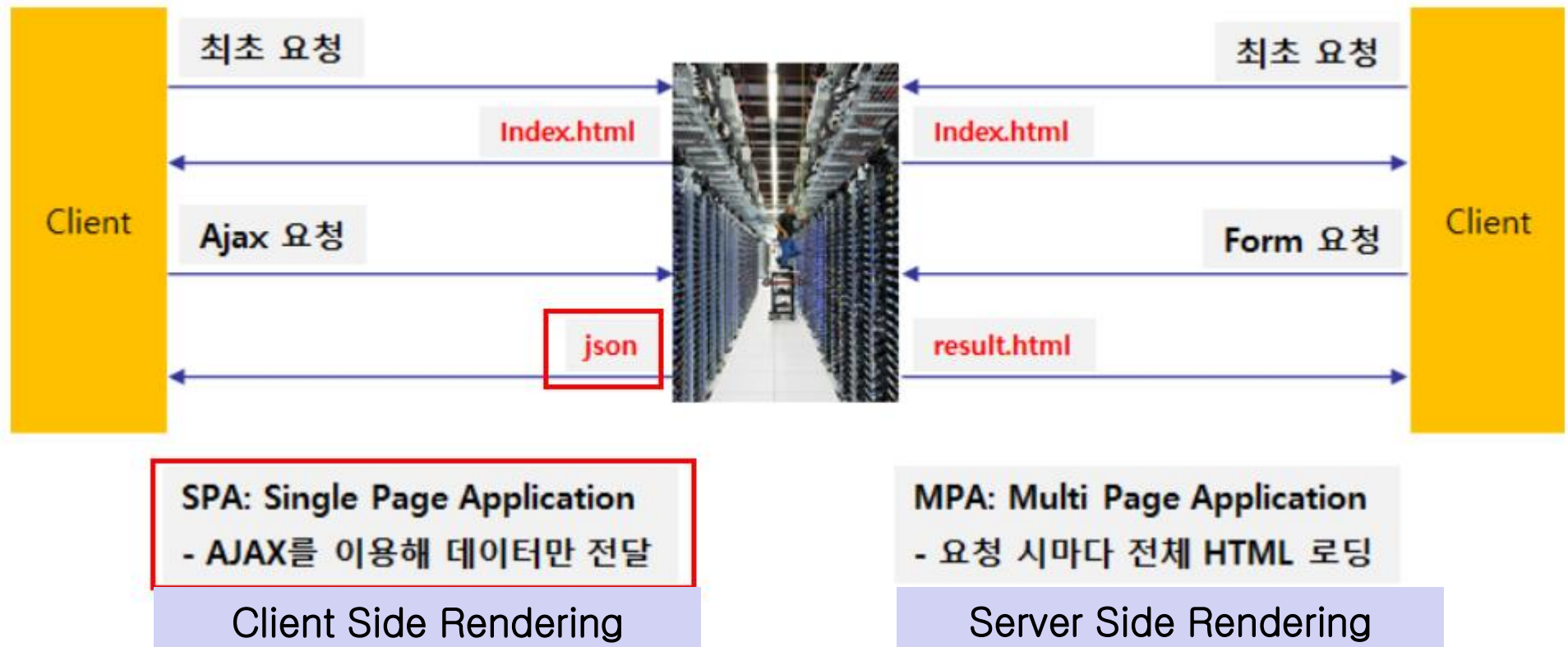
웹 어플리케이션의 UI 화면을 구현할 때 사용 가능한 형태는 SPA와 MPA가 있다.



## 2. SPA vs MPA 동작방식

Reactjs 18.2.0

SPA와 MPA의 동작 방식의 가장 큰 차이점은 일반적인 Form 전송이나 Ajax 동작이냐로 나뉘볼 수 있다.



SPA는 Single Page Application으로 Client Side Rendering을 추구한다.  
즉 UI화면과 관련된 리소스를 처음 요청시 서버로부터 몽땅 받아낸 후 클라이언트에서 모든 HTML/CSS/JS를 가지고 있다.  
이후 Ajax 통신을 통해 변경하고자 하는 데이터만 받아오게 된다.

#### 장점

SPA는 사용 중 리소스 로딩이 없기 때문에 부드럽게 화면 전환이 이루어진다.  
서버 입장에서는 템플릿(JSP)를 만드는 연산이 클라이언트로 분산되기 때문에 부담이 줄어든다.  
컴포넌트별로 개발하기 때문에 생산성이 향상된다.  
모바일 앱에서도 동일한 패턴의 Rest API 사용이 가능하다.

#### 단점

URL이 변경되지 않기 때문에 검색엔진의 색인화가 어렵다.  
초기 구동 비용이 MPA 대비 상대적으로 비싸다.

## 4. react-router 개요

Reactjs 18.2.0

<https://reactrouter.com/en/main>

### 1) react-router 설치 (v6.X 이상)

```
npm install react-router-dom
```

### 2) 실습 및 실행

#### App.js

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import HomePage from "../pages/Home";

const router = createBrowserRouter([
  { path: '/', element: <HomePage /> }
]);

function App() {
  return <RouterProvider router={router} />
}

export default App;
```

#### Home.js

```
export default function HomePage(){
  return (
    <>
    <h1>My Home Page</h1>
    </>
  )
}
```

## 5. Menu 역할 Root.js 추가

Reactjs 18.2.0

<https://reactrouter.com/en/main/start/tutorial#nested-routes>

```
1  const router = createBrowserRouter([
2    {
3      path: "/",
4      element: <Root />,
5      errorElement: <ErrorPage />,
6      children: [
7        {
8          path: "contacts/:contactId",
9          element: <Contact />,
10         },
11       ],
12     },
13   ]);
```

```
1  import { Outlet } from "react-router-dom";
2
3  export default function Root() {
4    return (
5      <>
6        {/* all the other elements */}
7        <div id="detail">
8          <Outlet />
9        </div>
10      </>
11    );
12  }
```

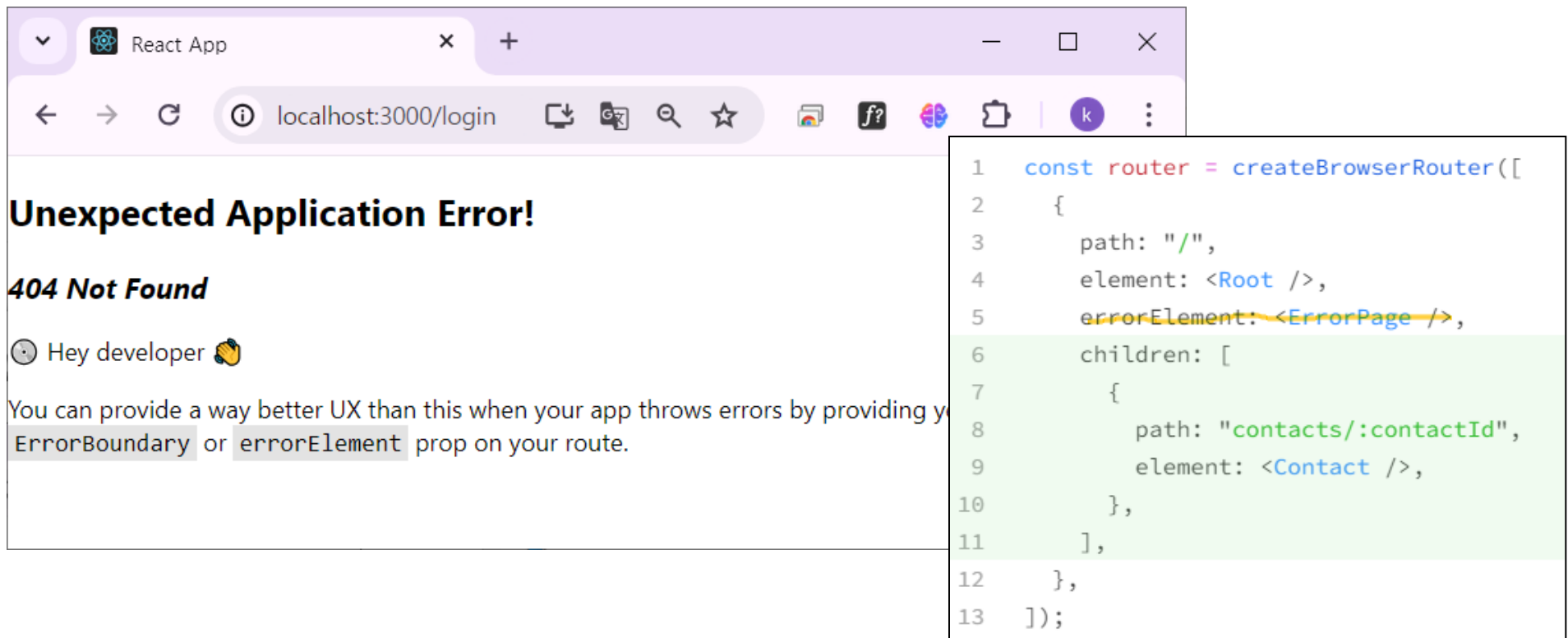
## 6. ErrorPage 추가

Reactjs 18.2.0

<https://reactrouter.com/en/main/start/tutorial#handling-not-found-errors>

<https://reactrouter.com/en/main/route/error-element#errorelement>

설정된 경로와 일치하지 않는 요청은 기본적으로 다음과 같이 에러 메시지가 출력된다. 이것을 ErrorPage를 작성하여 커스텀 예외처리 할 수 있다.



The image shows a web browser window with the address bar at `localhost:3000/login`. The page displays an "Unexpected Application Error!" message, followed by "404 Not Found" and a message from "Hey developer" suggesting the use of `ErrorBoundary` or `errorElement` prop. To the right, a code editor shows the following JavaScript code:

```
1  const router = createBrowserRouter([
2    {
3      path: "/",
4      element: <Root />,
5      errorElement: <ErrorPage />,
6      children: [
7        {
8          path: "contacts/:contactId",
9          element: <Contact />,
10         },
11       ],
12     },
13   ]);
```

## 6. ErrorPage 추가

Reactjs 18.2.0

ErrorPage.js

```
import { useRouteError } from "react-router-dom";

function ErrorPage() {
  const error = useRouteError();
  console.error(error);

  return (
    <>
      <main className='ErrorPage'>
        <p>Could not find this page!</p>
        <p>
          <i>{error.statusText || error.message}</i>
          <i>{error.data || error.message}</i>
        </p>
      </main>
    </>
  );
}

export default ErrorPage;
```



<https://reactrouter.com/en/main/components/nav-link#navlink>

<Link> 태그는 선택시 피드백이 없기 때문에 어떤 링크를 선택했는지 모른다.  
하지만 <NavLink> 태그는 자동으로 className(style) 속성의 함수에 isActive 속성이 전달되고 이 값을 활용하여 스타일을 지정할 수 있다.

```
1  import { NavLink } from "react-router-dom";
2
3  <NavLink
4    to="/messages"
5    className={({ isActive, isPending }) =>
6      isPending ? "pending" : isActive ? "active" : ""
7    }
8  >
9    Messages
10 </NavLink>;
```

<https://reactrouter.com/en/main/hooks/use-navigate#usenavigate>

useNavigate hook 을 활용하여 프로그래밍 방식의 라우팅 처리가 가능하다.

```
1  import { useNavigate } from "react-router-dom";
2
3  function useLogoutTimer() {
4    const userIsInactive = useFakeInactiveUser();
5    const navigate = useNavigate();
6
7    useEffect(() => {
8      if (userIsInactive) {
9        fake.logout();
10       navigate("/session-timed-out");
11     }
12   }, [userIsInactive]);
13 }
```

<https://reactrouter.com/en/main/hooks/use-params#useparams>

Spring의 REST 방식과 같이 URL에 파라미터를 포함시켜 서버에 전달할 수 있다.  
/경로/:id 형식으로 요청하고 useParams hook으로 전달된 파라미터 값을 얻는다.

```
1  import * as React from 'react';
2  import { Routes, Route, useParams } from 'react-router-dom';
3
4  function ProfilePage() {
5    // Get the userId param from the URL.
6    let { userId } = useParams();
7    // ...
8  }
9
10 function App() {
11   return (
12     <Routes>
13       <Route path="users">
14         <Route path=":userId" element={<ProfilePage />} />
15         <Route path="me" element={...} />
16       </Route>
17     </Routes>
18   );
19 }
```

<https://reactrouter.com/en/main/components/link#relative>

경로지정시 / 로 시작하면 절대 경로이고 아니면 상대 경로이다.

<Link> 를 이용한 상대 경로 지정시 `relative="route|path"` 속성값을 사용할 수 있다.  
기본값은 `route` 이다.

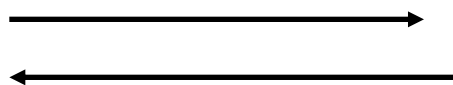
```
1  // Contact and EditContact do not share additional UI layout
2  <Route path="/" element={<Layout />}>
3    <Route path="contacts/:id" element={<Contact />} />
4    <Route
5      path="contacts/:id/edit"
6      element={<EditContact />}
7    />
8  </Route>;
9
10 function EditContact() {
11   // Since Contact is not a parent of EditContact we need to go up one level
12   // in the current contextual route path, instead of one level in the Route
13   // hierarchy
14   return (
15     <Link to=".." relative="path">
16       Cancel
17     </Link>
18   );
19 }
```

## 동작방식

```
{path: '/products/:productId', element: <ProductDetailPage />
```

### Products 화면

/products/2  
/

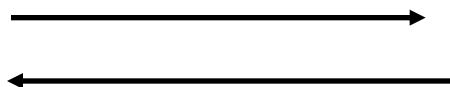


### ProductDetailPage 화면

<Link to=".." relative="route">back</Link>

route로 지정하면 /products/2 가 하나의 route 경로이기 때문에 상위 경로로 가면 / 가 된다.

/products/2  
/products



<Link to=".." relative="path">back</Link>

path로 지정하면 /products/2 는 두개의 path 경로이기 때문에 상위 경로가 가면 /products 가 된다.

## 11장. 라우팅 심화 기능

# 1. loader 이용한 데이터 fetching

Reactjs 18.2.0

<https://reactrouter.com/en/main/route/loader#loader>

loader는 링크를 통한 컴포넌트가 생성되기 전에 실행되어 컴포넌트에 데이터를 전달하는 역할을 수행할 수 있다.

요청 받은 컴포넌트 및 하위 컴포넌트에서 useLoaderData() hook으로 loader가 return 한 데이터를 사용할 수 있다.

```
1  createBrowserRouter([
2    {
3      element: <Teams />,
4      path: "teams",
5      loader: async () => {
6        return fakeDb.from("teams").select("*");
7      },
8      children: [
9        {
10         element: <Team />,
11         path: ":teamId",
12         loader: async ({ params }) => {
13           return fetch(`/api/teams/${params.teamId}.json`);
14         },
15       },
16     ],
17   },
18 ]);
```

# 1. loader 이용한 데이터 fetching

Reactjs 18.2.0

## App.js 수정

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <RootLayout />,
    errorElement: <ErrorPage />,
    children: [
      {path: '/', element: <HomePage />},
      {path: '/products', element: <ProductsPage />},
      {path: '/products/:productId', element: <ProductDetailPage /> },
      {path: '/users', element: <UsersPage />,
        loader: async function () {
          console.log("loader>>>>>>>>>")
          const response = await fetch('https://reqres.in/api/users?page=2');
          const resData = await response.json();
          return resData.data;
        }
      }
    ]
  }
])
```

## Users.js 추가

```
import UsersList from '../components/UsersList';
import { useLoaderData } from 'react-router-dom';

function UsersPage() {
  const events = useLoaderData();
  return(
    <>
      <h1>The Users Page</h1>
      <UsersList events={events}/>
    </>
  )
}
export default UsersPage;
```

## UsersList.js 추가

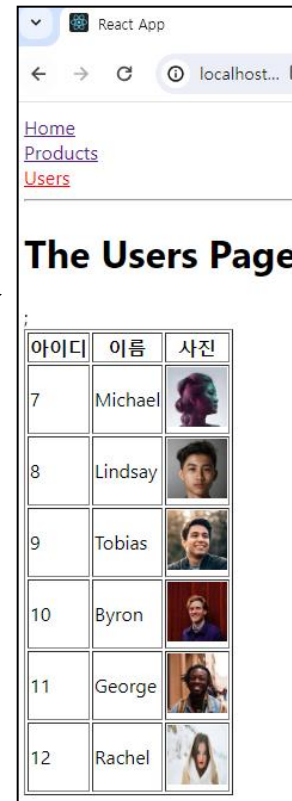
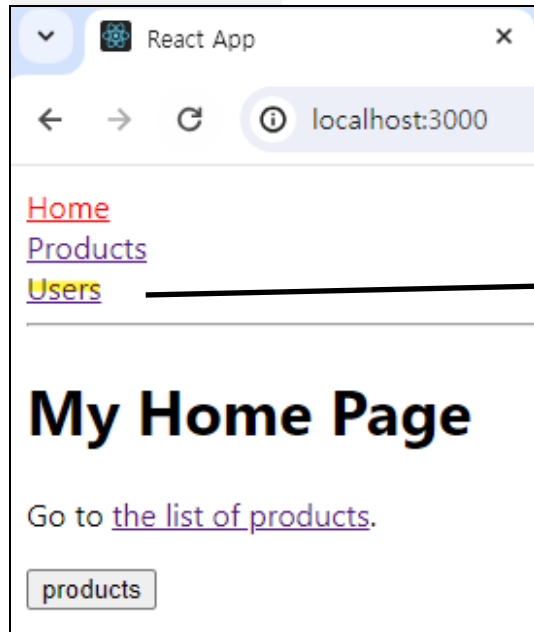
```
export default function UsersList({events}){
  return(
    <>
      <table border="1">
        <thead>
          <tr>
            <th>아이디</th>
            <th>이름</th>
            <th>사진</th>
          </tr>
        </thead>
        <tbody>
          {
            events.map((user)=>(
              <tr key={user.id}>
                <td>{user.id}</td>
                <td>{user.first_name}</td>
                <td><img src={user.avatar} width={50} height={50} /></td>
              </tr>
            ))
          }
        </tbody>
      </table>
    </>
  )
}
```



# 1. loader 이용한 데이터 fetching

Reactjs 18.2.0

## 실행결과



## 2. params 속성 vs request 속성

Reactjs 18.2.0

### params 속성

```
1  createBrowserRouter([
2    {
3      path: "/teams/:teamId",
4      loader: ({ params }) => {
5        return fakeGetTeam(params.teamId);
6      },
7    },
8  ]);
```

### request 속성

```
1  function loader({ request }) {
2    const url = new URL(request.url);
3    const searchTerm = url.searchParams.get("q");
4    return searchProducts(searchTerm);
5  }
```

### 3. Id 이용한 선택적 loader 사용

Reactjs 18.2.0

<https://reactrouter.com/en/main/hooks/use-route-loader-data#userouteloaderdata>

```
1  createBrowserRouter([
2    {
3      path: "/",
4      loader: () => fetchUser(),
5      element: <Root />,
6      id: "root",
7      children: [
8        {
9          path: "jobs/:jobId",
10         loader: loadJob,
11         element: <JobListing />,
12       },
13     ],
14   },
15 ]);
```

```
const user = useRouteLoaderData("root");
```

## 4. useNavigation

Reactjs 18.2.0

<https://reactrouter.com/en/main/hooks/use-navigation#usenavigation>

이전에는 isFetching state 값을 직접 구현하여 사용자가 요청한 이벤트의 활성화 관련 피드백을 제공함.

useNavigation hook을 사용하면 현재 경로 전환 상태를 매우 쉽게 알 수 있다.

즉 전환이 시작됐고 데이터가 도착하길 기다리는 중인지 아니면 완료됐는지 상태를 알 수 있게 된다.

```
1  import { useNavigation } from "react-router-dom";
2
3  function SomeComponent() {
4    const navigation = useNavigation();
5    navigation.state;
6    navigation.location;
7    navigation.formData;
8    navigation.json;
9    navigation.text;
10   navigation.formAction;
11   navigation.formMethod;
12   navigation.formEncType;
13 }
```

```
1  function SubmitButton() {
2    const navigation = useNavigation();
3
4    const text =
5      navigation.state === "submitting"
6        ? "Saving..."
7        : navigation.state === "loading"
8          ? "Saved!"
9          : "Go";
10
11   return <button type="submit">{text}</button>;
12 }
```

## 4. useNavigation

Reactjs 18.2.0

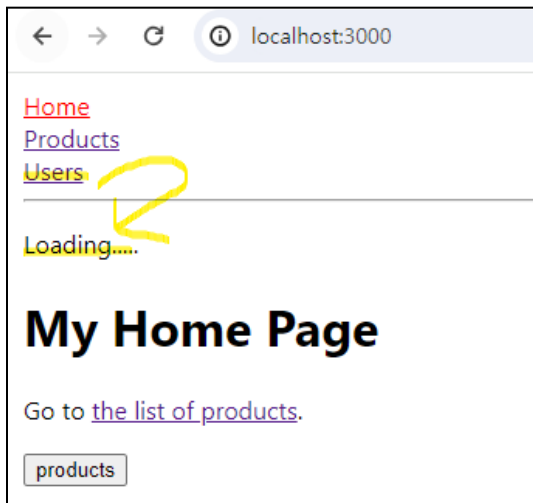
이전에는 isFetching state 값을 직접 구현하여 사용자가 요청한 이벤트의 활성화 관련 피드백을 제공함.

대신 useNavigation hook을 사용하면 현재 경로 전환 상태를 매우 쉽게 알 수 있다. 즉 전환이 시작됐고 데이터가 도착하길 기다리는 중인지 아니면 완료됐는지 상태를 알 수 있게 된다.

**문법**

```
const navigation = useNavigation();  
..  
navigation.state="idle|loading|submitting";
```

### 실행결과



### Root.js 코드 추가

```
export default function RootLayout(){  
  const navigation = useNavigation();  
  return(  
    <div style={{margin:'10px'}}>  
      <MainNavigation />  
      <hr/>  
      <main>  
        {navigation.state==='loading' && <p>Loading.....</p>}  
        <Outlet />  
      </main>  
    </div>  
  )  
}
```

## 5. loader 함수 예외처리

Reactjs 18.2.0

<https://reactrouter.com/en/main/route/error-element#throwing-responses>

```
import { json } from "react-router-dom";

function loader() {
  const stillWorksHere = await userStillWorksHere();
  if (!stillWorksHere) {
    throw json(
      {
        sorry: "You have been fired.",
        hrEmail: "hr@bigco.com",
      },
      { status: 401 }
    );
  }
}
```

```
function ErrorBoundary() {
  const error = useRouteError();

  if (isRouteErrorResponse(error) && error.status === 401) {
    // the response json is automatically parsed to
    // `error.data`, you also have access to the status
    return (
      <div>
        <h1>{error.status}</h1>
        <h2>{error.data.sorry}</h2>
        <p>
          Go ahead and email {error.data.hrEmail} if you
          feel like this is a mistake.
        </p>
      </div>
    );
  }
}
```

## 6. action 함수 이용한 Form 전송

Reactjs 18.2.0

<https://reactrouter.com/en/main/route/action#action>

loader() 함수를 활용하여 컴포넌트가 생성되기 전에 필요한 데이터를 얻을 수 있듯이 action() 함수를 활용하면 <Form method="post"> 태그의 데이터를 서버에 전송할 수 있다. 이때 폼은 리액트에서 제공한 <Form> 이어야 되고 post로 지정해야 된다. 요청한 폼 데이터는 action({request}) 의 request.formData() 로 얻을 수 있다.

```
1 <Route
2   path="/song/:songId/edit"
3   element={<EditSong />}
4   action={async ({ params, request }) => {
5     let formData = await request.formData();
6     return fakeUpdateSong(params.songId, formData);
7   }}
8   loader={({ params }) => {
9     return fakeGetSong(params.songId);
10  }}
11 />
```

```
1 <Form method="post">
2   <input name="songTitle" />
3   <textarea name="lyrics" />
4   <button type="submit">Save</button>
5 </Form>;
6
7 // accessed by the same names
8 formData.get("songTitle");
9 formData.get("lyrics");
```

## 6. action 함수 이용한 Form 전송

Reactjs 18.2.0

### NewUsers.js 코드 작성

```
import {
  Form,
  useNavigate,
  json,
  redirect } from 'react-router-dom';
export default function NewUsersPage() {
  const navigate = useNavigate();
  function cancelHandler() {
    navigate('..');
  }
  return(
    <>
    <h1>The NewUsersPage Page</h1>
    <Form method='post'>
      id:<input type="text" name="id" /><br/>
      email:<input type="text" name="email" /><br/>
      first_name:<input type="text" name="first_name" /><br/>
      last_name:<input type="text" name="last_name" /><br/>
      <button type="button" onClick={cancelHandler}>
        Cancel
      </button>
      <button>Save</button>
    </Form>
    </>
  )
}
```

```
export async function action({ request, params }) {
  const data = await request.formData();

  const eventData = {
    id: data.get('id'),
    email: data.get('email'),
    first_name: data.get('first_name'),
    last_name: data.get('last_name'),
  };

  const response = await fetch('https://reqres.in/api/users', {
    method: 'POST',
    body: JSON.stringify( eventData ),
    headers: {
      'Content-Type': 'application/json',
    },
  });

  if (!response.ok) {
    throw json({ message: 'Could not save user.' }, { status: 500 });
  }
  // 실습서버인 reqres.in 에 잘 전송되었는지 확인용
  const resData = await response.json();
  console.log("resData:", resData);

  return redirect('/users');
}
```



## 12장. CSS 적용

## 개요

React 공식 문서에서는 명확한 스타일링 가이드를 제공하지 않는다.  
워낙 다양한 방법이 있기 때문인 것으로 판단된다.

## 적용방법

- 가. Inline Style
- 나. External Style
- 다. CSS modules
- 라. CSS-in JS ( Styled Components)

### 개요

가장 간단하고 쉬운 방법은 해당 React 컴포넌트에 CSS 인라인 스타일(inline style)을 바로 적용하는 것이다. 일반적인 웹페이지에서 적용하는 것처럼 html의 style 속성을 이용한다. 하지만 React는 자바스크립트로 작성하기 때문에 일반 웹페이지에서 적용할 때와 차이점이 있다.

가. style 속성값에 일반 문자열이 아닌 자바스크립트 객체가 할당된다.

나. css 속성명은 케밥 케이스(kebab case)가 아닌 카멜케이드(camel case)로 작성한다.

### 샘플

```
import React from "react";

const btnStyle = {
  color: "white",
  background: "teal",
  padding: ".375rem .75rem",
  border: "1px solid teal",
  borderRadius: ".25rem",
  fontSize: "1rem",
  lineHeight: 1.5,
};

function Button() {
  return <button style={btnStyle}>Inline</button>;
}
```

### 개요

별도의 파일에 스타일을 정의해 놓고 React 컴포넌트 파일에서 해당 css 파일을 import 한다. 그 다음 엘리먼트의 className 속성을 이용해서 외부 파일에 정의된 스타일을 맵핑 시켜주는 방법이다.

### 샘플

#### Button.css

```
.btn {  
  color: white;  
  background: teal;  
  padding: 0.375rem 0.75rem;  
  border: 1px solid teal;  
  border-radius: 0.25rem;  
  font-size: 1rem;  
  line-height: 1.5;  
}
```

```
import React from "react";  
import "./Button.css";  
  
function Button() {  
  return <button className="btn">External</button>;  
}
```

### 개요

external style 방식은 React 앱의 규모가 커짐에 따라 CSS 이름이 겹치게 될 가능성이 매우 커진다.

이 문제를 해결하기 위한 방법으로 각 CSS 파일에 고유한 네임스페이스를 부여해주는 CSS 모듈(CSS Modules) 방법을 사용할 수 있다.

이렇게 CSS 모듈을 사용하면 각 CSS 파일마다 고유한 네임스페이스를 부여해주기 때문에 각 React 컴포넌트는 완전히 격리된 스타일을 보장 받을 수 있다.

### 적용방법

가. external style 작성시 .css 확장자가 아닌 .module.css 확장자 사용.

나. import 할 때 CSS 모듈의 이름을 명시적으로 지정.

예> `import module_name from './xxx.module.css';`

다. className 속성으로 css 설정시 CSS모듈명을 지정한다.

예> `className={module_name.class_name}`

## 4. CSS-in-JS ( Styled Component)

Reactjs 18.2.0

### 개요

CSS in JS는 스타일 정의를 CSS 파일이 아닌 자바스크립트로 작성된 컴포넌트에 정의하고 사용하는 스타일 기법이다.

과거에는 html, css, javascript는 각각 별도의 파일에 저장하고 사용하는 방법이 best practice 였으나 최근에는 웹 어플리케이션을 여러 개의 재활용이 가능한 빌딩 블록으로 개발하는 컴포넌트 기반 개발 방법이 대세이다.

즉 개별적인 컴포넌트에 html, css, javascript를 모두 포함하는 방법이다.

### 패키지 설치

```
npm install styled-components
```

### html 태그에 적용

```
import styled from "styled-components";

// 1. html 태그에 스타일 적용 ==> styled.태그명`` 형식
const StyledButton = styled.button`
  color:blue;
  background-color: yellow;
  font-size: 16px;
`;
const StyledH1 = styled.h1`
  color:red;
`;
```

```
function App() {
  return (
    <div>
      <StyledH1>Hello</StyledH1>
      <StyledButton>syteled components</StyledButton>
    </div>
  );
}

export default App;
```

## 13장. 성능 최적화 기법

## 개요

부모와 자식 간의 props 전달시 사용되는 함수 컴포넌트를 메모이제이션하는 방법이다.

기본적으로 부모가 재랜더링 되면 자식도 재랜더링이 된다.

이때 부모가 재랜더링 되더라도 자식에게 매번 동일한 props를 전달하는 경우에는 memo 함수를 사용하여 자식의 재랜더링을 방지할 수 있다.

## 문법

```
import {memo} from 'react';  
const Counter = memo(function ({props}){});
```

App이 재랜더링되면 Counter도 재랜더링됨.

App -----> Counter({initialCount})

App이 재랜더링되도 props가 동일하면 Counter 재랜더링 안됨.

App -----> memo(Counter({initialCount}))



## 2. useMemo hook

Reactjs 18.2.0

<https://ko.legacy.reactjs.org/docs/hooks-reference.html#usememo>

<https://react.dev/reference/react/useMemo#usage>

### 개요

컴포넌트의 성능을 최적화 시킬 수 있는 대표적인 hook 중 하나로서 기존에 수행한 복잡한 연산의 결과값을 저장해 두고 동일한 입력이 들어오면 재활용하는 용도로 사용한다.

### 문법

```
const value = useMemo( function() {}, [의존성값])
```

useMemo는 첫번째 인자는 콜백함수를 두번째 인자는 의존성 배열을 지정한다. 두번째 인자인 배열의 요소값이 업데이트될 때만 콜백함수를 다시 호출한다. 만약 빈 배열([])을 지정하면 맨 처음 컴포넌트가 마운트 되었을 때만 값을 계산하고 이후에는 항상 memoization된 값을 반환한다. value에 저장된 값은 콜백함수가 리턴 시킨 결과값이다.

### 3. useCallback hook

Reactjs 18.2.0

<https://ko.reactjs.org/docs/hooks-reference.html#usecallback>

#### 개요

컴포넌트가 재렌더링 될 때마다 내부적으로 사용된 함수가 새롭게 생성된다. 이렇게 불필요하게 매번 생성되는 함수를 재생성 되지 않도록 방지할 수 있다.

#### 문법

```
import {useCallback} from 'react';  
const fun = useCallback( function (){}, []);
```

만약에 익명함수안에서 사용하는 state 또는 props가 있다면 [] 배열안에 꼭 포함시켜야 된다. 포함하지 않으면 해당 값을 참조할 때 가장 최신 값을 참조할 것이라 보장할 수 없다.

<https://ko.reactjs.org/docs/hooks-reference.html#usereducer>

#### 개요

useState 혹은 동일하게 상태를 관리할 때 사용한다.  
하지만 컴포넌트와 상태 업데이트 로직을 분리할 수 있도록 구현이 가능하다.  
또한 useState 와 다르게 데이터 변경흐름이 단방향으로 진행되어 직관적이고 예측이 가능한 장점이 있다. 이벤트가 발생하면 dispatch 함수에 의해 trigger 된다.

#### 구현

##### 1. import

```
import { useReducer } from 'react';
```

##### 2. action 형식

형식1: { type:'INCREMENT' }

형식2: { type:'CHANGE\_INPUT', key:'email', value:'zzzz@daum.net' }

..

### 3. reducer

```
const reducer = (state, action) => {  
  switch(action.type){  
    case "INCREMENT": return state+1;  
    case "DECREMENT": return state-1;  
    default: return state;  
  }  
}
```

### 4. reducer 생성

```
const [state, dispatch] = useReducer(reducer, initialState);
```

### 5. 이벤트 발생

```
const onIncrease = () => {  
  dispatch({ type: 'INCREMENT' });  
};
```



수고하셨습니다.