# 리액트 프로그래밍

## (React Programming)

# 1장 리액트 시작

리액트는 자바스크립트 라이브러리로 사용자 인터페이스를 만드는데 사용한다. 구조가 MVC 프레임워크와 달리 오직 View만 신경 쓰는 라이브러리이다. 리액트 프로젝트에서 특정 부분이 어떻게 생길지 정하는 선언체가 있는데 이를 컴포넌트라고 한다. 컴포넌트는 재사용이 가능한 API로 수많은 기능들을 내장하고 있으며, 컴포넌트 하나에서 해당 컴포넌트의 생김새와 작동 방식을 정의한다.

- 리액트의 특징

리액트는 Virtual DOM 방식을 사용하여 DOM 업데이트를 추상화함으로써 DOM 처리 횟수를 최소화하고 효율적으로 진행한다. 리액트에서 데이터가 변하여 웹 브라우저에서 실제 DOM을 업데이트할 때는 다음 세 가지 절차를 밟는다.

1. 데이터를 업데이트 하면 전체 UI를 Virtual DOM에 리렌더링 한다.
2. 이전 Virtual DOM에 있던 내용과 현재 내용을 비교한다.
3. 바뀐 부분만 실제 DOM에 적용한다.

- 작업 환경 설정

1. Node.js와 npm

리액트 프로젝트를 만들 때는 Node.js를 반드시 먼저 설치한다. Node.js는 크롬 V8 자바 스크립트 엔진으로 빌드 한 자바스크립트 런타임이다. 이것으로 웹 브라우저가 아닌 곳에서도 자바스크립트를 사용하여 연산할 수 있다.

리액트 애플리케이션은 웹 브라우저에서 실행되는 코드이므로 Node.js와 직접적인 연관은 없지만 프로젝트를 개발하는데 필요한 주요 도구들이 Node.js를 사용하기 때문에 설치하는 것이다. 이때 사용되는 도구에는 ECMAScript 6(2015년 공식적으로 업데이트한 자바스크립트 문법)를 호환시켜주는 바벨(babel), 모듈화 된 코드를 한 파일일로 합치고(번들링) 코드를 수정할 때마다 웹브라우저를 리로딩하는 등의 여러 기능을 지닌 웹팩(webpack)등이 있다.

Node.js를 설치하면 Node.js 패키지 매니저 도구인 npm이 설치된다. npm으로 수많은 개발자가 만든 패키지(재사용 가능한 코드)를 설치하고 설치한 패키지의 버전을 관리할 수 있다. 리액트 역시 하나의 패키지이다.

2. Node.js와 npm 설치

Node.js 공식 내려 받기 페이지(https://nodejs.org/ko/download)에서 Windows Installer를 내려 받아 설치한다. 설치가 끝나면 터미널(또는 명령 프롬프트) 창을 열고 아래 명령어를 입력하여 제대로 설치했는지 확인한다.

```
node -v
```

3. yarn 설치 (npm install -global yarn)

Node.js를 설치할 때 패키지를 관리해 주는 도구가 npm이다. yarn은 npm을 대체할 수 있는 도구로 npm보다 더 빠르며 효율적인 캐시시스템과 기타 부가기능을 제공한다. yarn 내려 받기 페이지(https://yarnpkg.com/en/docs/install#widows-stable)에서 Download Installer 버튼을 눌러 설치 프로그램을 내려 받은 후 실행한다. 터미널을 열어 yarn을 제대로 설치했는지 확인한다.

```
yarn --version
```

4. 에디터 설치

VS Code 공식 내려 받기 페이지(https://code.visualstudio.com/Download)에서 운영체제에 맞는 버전을 설치한다.

5. VS Code 확장 프로그램 설치

아래 프로그램들은 VS Code를 사용할 때 설치하면 유용한 확장 프로그램들이다.

- ESLint: 자바스크립트 문법 및 코드 스타일을 검사해 주는 도구이다.
- Reactjs Code Snippet: 리액트 컴포넌트 및 라이프사이클 함수를 작성할 때 단축 단어를 사용하여 간편하게 코드를 자동으로 생성해 낼 수 있는 코드 스니펫 모음이다. 검색했을 때 유사한 결과가 여러 개 나올 수 있는데 제작자가 charalampos karypidis인 것을 설치한다.
- Prettier-Code formatter: 코드 스타일을 자동으로 정리해 주는 도구이다.

- create-react-app으로 프로젝트 생성하기

create-react-app은 리액트 프로젝트를 생성할 때 필요한 웹팩, 바벨의 설치 및 설정 과정을 생략하고 바로 간편하게 프로젝트 작업 환경을 구축해 주는 도구이다. 아래 명령을 실행해 프로젝트를 생성하고 프로젝트 디렉터리로 이동한 뒤 리액트 개발 전용 서버를 구동해 본다.

```
yarn create react-app ex01  yarn을 사용하지 않는 경우 npm init react-app <프로젝트명>
cd ex01
yarn start
```

# 2장 JSX

- JSX란?

JSX는 자바스크립트의 확장 문법이며 XML과 매우 비슷하게 생겼다. JSX로 작성한 코드는 브라우저에서 실행되기 전에 코드가 번들링되는 과정에서 바벨을 사용하여 일반 자바스크립트 코드로 변환된다. 바벨에서는 여러 문법을 지원할 수 있도록 preset 및 plugin을 설정한다.

- JSX의 문법

### 1. 감싸인 요소

컴포넌트에 여러 요소가 있다면 반드시 부모 요소 하나로 감싸야 한다. Virtual DOM에서 컴포넌트 변화를 감지해 낼 때 효율적으로 비교할 수 있도록 컴포넌트 내부는 하나의 DOM 트리 구조로 이루어져야 한다는 규칙이 있기 때문이다. return문 안에서 JSX문법을 사용한다.

[src]-[App.js]

```
function App() {
    return(
       <div>
           <h1>리액트 안녕!</h1>
           <h2>잘 작동하나?</h2>
       </div>
    )
}
export default App;
```

### 2. 자바스크립트 표현

JSX 안에서는 자바스크립트 표현식을 쓸 수 있다. 자바스크립트 표현식을 작성하려면 JSX 내부에서 코드를 { }로 감싸준다.

[src]-[App.js]

```
function App() {
    const name = '리액트';  //const는 ES6문법에서 새로 도입되었고 변경이 불가능한 상수를 선언 let은 동적인 값을 담을 수 있는 변수를 선언
    return(                 //var 키워드는 scope가 함수 단위이고 const와 let은 블록단위이다.
        <div
            //주석 작성방법1: 시작 태그를 여러 줄로 작성하게 된다면 여기에 주석을 작성할 수 있다.
        >
            <h1>{ name } 안녕!</h1>  {/* 주석 작성방법2: JSX에서 주석은 이렇게 사용 합니다 */}
            <h2>잘 작동하나?</h2>
        </div>
    )
}
```

### 3. If문 대신 조건부 연산자

[src]-[App.js]

```
function App() {
  const name = '리액트';
  return(
    <div>
        { name === '리액트' ? <h1>리액트입니다.</h1> : <h1>리액트가 아닙니다.</h1> }
    </div>
  )
}
```

### 4. AND 연산자(&&)를 사용한 조건부 렌더링 : 만족하지 않을 때 아예 아무것도 렌더링하지 않아야 하는 상황에 사용

[src]-[App.js]

```
return <div>{ name === '리액트' && <h1>리액트 입니다.</h1> }</div>
```

### 5. OR 연산자( || )를 사용한 조건부 렌더링 : undefined일 때 사용할 값을 지정해야 할 상황에 사용

[src]-[App.js]

```
const name = undefined;
return <div>{ name || <h1>리액트 입니다.</h1> }</div>
```

### 6. 인라인 스타일링

리액트에서 스타일을 적용할 때는 문자열 형태로 넣는 것이 아니라 객체 형태로 넣어 준다. 스타일 이름 중에서 background-color처럼 - 문자가 포함되는 이름은 - 문자를 없애고 카멜 표기법으로 작성해야한다. 따라서 background-color는 backgroundColor로 작성한다.

[src]-[App.js]

```
function App() {
    const name = '리액트';
    const style={
        backgroundColor: 'black',
        color: 'aqua',
        fontSize: '48px',
        fontWeight: 'blold',
        padding: 16  // 단위를 생략하면 px로 지정된다.
    }
    return (
        <div style={style}>
            {name}
        </div>
    );
}
export default App;
```

위 예제는 style 객체를 미리 선언하고 div의 style 값으로 지정해 주었는데 아래는 미리 선언하지 않고 바로 style 값을 지정하였다.

[src]-[App.js]

```
function App() {
    const name = '리액트';
    return (
        <div
            style={{
                backgroundColor: 'black',
                color: 'aqua',
                fontSize: '48px',
                fontWeight: 'blold',
                padding: 16
            }}>
            {name}
        </div>
    );
}
export default App;
```

### 7. class 대신 className

일반 HTML에서 CSS 클래스를 사용할 때는 와 같이 class라는 속성을 설정한다. 하지만 JSX에서는 class가 아닌 className으로 설정해 주어야 한다.

[src]-[App.css]

```
.react {
    background: aqua;
    color: black;
    font-size: 48px;
    font-weight: bold;
    padding: 16px;
}
```

[src]-[App.js]

```
import './App.css';

function App() {
    const name = '리액트';
    return (
        <div className='react'>{name}</div>
    );
}
export default App;
```

# 3장 컴포넌트

**리액트를 사용하여 애플리케이션의 인터페이스를 설계할 때 사용자가 볼 수 있는 요소는 여러 가지 컴포넌트로 구성되어있다. 컴포넌트의 기능은 단순한 템플릿 이상이다. 데이터가 주어졌을 때 이에 맞추어 UI를 만들어 주는 것은 물론이고 라이프사이클 API를 이용하여 컴포넌트가 화면에서 나타날 때, 사라질 때, 변화가 일어날 때 주어진 작업들을 처리할 수 있으며 임의 메서드를 만들어 특별한 기능을 붙여 줄 수 있다.**

- **자동완성: Extensions에서 ES7+ React/Redux/React-Native snippets 설치**

- **App 컴포넌트 구성**

**[public]-[index.html]**

```
<html lang="en">
    <head> ...</head>
    <body>
        <div id="root"></div>
    </body>
</html>
```

**[src]-[index.js]**

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById('root')
);
reportWebVitals();
```

**[src]-[index.css]**

```
body {
    margin: 0;
    padding: 0;
    background: #e9ecef;
}
```

**[src]-[App.js]**

```
import './App.css';

function App() {
    const name = 'Tom';
    const naver = {
        name: '네이버',
        url: 'https://naver.com'
    };
    return (
        <div className="App">
            <h1 style={{ color: "yellow", background: "green" }}>
                Hello, { name }
            </h1>
            <p>{ 2 + 5 }</p>
            <a href ={ naver.url }>{ naver.name }></a>
        </div>
    );
}
export default App;
```

**[src]-[App.css]**

```
.App {
    text-align: center;
}
```

- **Hello, Welcome, World, App 함수형 컴포넌트 작성**

1) [src]-[World.js]

```
export default function World(){
    return <h3>World</h3>
};
```

2) [src]-[Hello.js]

```
import World from "./Word"

const Hello = () => {
    return(
        <div>
            <h1>Hello</h1>
            <World/>
        </div>
    );
};

export default Hello;
```

3) [src]-[Welcome.js]

```
export default function Welcome(){
    return <h1>Welcome</h1>
}
```

4) [src]-[App.js]

```
import './App.css';
import Hello from './Hello';
import Welcome from './Welcome';

function App() {
    return (
        <div className="App">
            <Hello/>
            <Welcome/>
            <Hello/>
        </div>
    );
}

export default App;
```

- **App 클래스형 컴포넌트**

[src]-[App.js]

```
import { Component } from 'react';
import './App.css';
import Hello from './Hello';
import Welcome from './Welcome';

class App extends Component {
    render(){
        return (
            <div className="App">
                <Hello/>
                <Welcome/>
                <Hello/>
            </div>
        );
    }
}

export default App;
```

- **props**

**props는 properties를 줄인 말로 컴포넌트 속성을 설정할 때 사용하는 요소이다. props는 컴포넌트가 사용되는 과정에서 부모 컴포넌트가 설정하는 값이며 컴포넌트 자신은 해당 props를 읽기 전용으로만 사용할 수 있다. props를 바꾸려면 부모 컴포넌트에서 바꾸어 주어야 한다.**

- **컴포넌트를 사용할 때 props 값 지정하기**

[src]-[MyComponent.js] 단축키 rc(react component)

```
const MyComponent = props => {
    return (
        <div>
            <h1>안녕하세요! 제 이름은 { props.name } 입니다.</h1>
            <h1>children 값은 { props.children } 입니다.</h1>
        </div>
    );
};

MyComponent.defaultProps = {
    name: '무기명'
};

export default MyComponent;
```

[src]-[App.js]

```
import MyComponent from './MyComponent';

const App = () => {
    return(
        <div>
            <MyComponent name="홍길동"/>
            <MyComponent/>
            <MyComponent>컴포넌트 자식</MyComponent>
        </div>
    );
};

export default App;
```

- **비구조화 할당 문법을 통해 props 내부 값 추출하기**

[src]-[App.js]

```
const MyComponent = props => {
    const { name, children } = props;
    return (
        <div>
            <h1>안녕하세요! 제 이름은 { name }입니다.</h1>
            <h1>children 값은 { children }입니다.</h1>
        </div>
    );
};

export default MyComponent;
```

[src]-[App.js]

```
const MyComponent = ({ name, children }) => {
    return (
        <div>
            <h1>안녕하세요! 제 이름은 { name } 입니다.</h1>
            <h1>children 값은 { children } 입니다.</h1>
        </div>
    );
};

export default App;
```

- **propTypes를 통한 props 검증**

[src]-[MyComponent.js]

```
import PropTypes from 'prop-types';

const MyComponent = ({ name, children }) => {
    return (
        <div>
            <h1>안녕하세요! 제 이름은 { name }입니다.</h1>  {/* App.js에서 name값을 { 1 } 숫자로 주면 Console창에 오류가 난다. */}
            <h1>children 값은 { children } 입니다.</h1>
        </div>
    );
}

MyComponent.propTypes = {
    name: PropTypes.string
}

export default MyComponent;
```

- **isRequired를 사용하여 필수 propTypes 설정**

[src]-[App.js]

```
import PropTypes from 'prop-types';

const MyComponent = ({ name, children, favoriteNumber }) => {
    return (
        <div>
            <h1>안녕하세요! 제 이름은 { name }입니다.<h1>
            <h1>children 값은 { children }입니다.<h1>
            <h1>제가 좋아하는 숫자는 { favoriteNumber }입니다.<h1>
        </div>
    );

}

MyComponent.defaultProps = {
    name: '무기명'
}

MyComponent.propTypes = {
    name: PropTypes.string,
    favoriteNumber: PropTypes.number.isRequired  //App.js에서 favoriteNumber를 설정하지 않으면 경고 발생
}

export default MyComponent;
```

- **클래스형 컴포넌트에서 props 사용하기**

[src]-[MyComponent.js]

```
import PropTypes from 'prop-types';
import { Component } from 'react';

class MyComponent extends Component {
    render(){
        const { name, favoriteNumber, children } = this.props;  //비구조화 할당
        return (
            <div>
                <h1>안녕하세요! 제 이름은 { name }입니다.<h1>
                <h1>children 값은 { children }입니다.<h1>
                <h1>제가 좋아하는 숫자는 { favoriteNumber } 입니다.</h1>
            </div>
        );
    }
}
...
export default MyComponent;
```

- **클래스 내부에 defaultProps와 propTypes 설정**

[src]-[MyComponent.js]

```
import PropTypes from 'prop-types';
import { Component } from 'react';

class MyComponent extends Component {
    static defaultProps = {
        name: '기본값'
    };

    static propTypes = {
        name: PropTypes.string,
        favoriteNumber: PropTypes.number.isRequired
    }

    render(){
        const { name, favoriteNumber, children } = this.props;  //비구조화 할당
        return (
            ...
        );
    }
}

export default MyComponent;
```

- **state**

**리액트에서 state는 컴포넌트 내부에서 바뀔 수 있는 값을 의미한다. 리액트에는 두 가지 종류의 state가 있다. 하나는 클래스형 컴포넌트가 지니고 있는 state이고 다른 하나는 함수형 컴포넌트에서 useState라는 함수를 통해 사용하는 state이다.**

- **클래스형 컴포넌트의 state**

[src]-[Counter.js]

```
import { Component } from "react";

class Counter extends Component{
    constructor(props){
        //현재 클래스형 컴포넌트가 상속받고 있는 생성자 함수를 호출
        super(props);
        //state의 초깃값 설정
        this.state = {
            number: 0
        };
    }

    render(){
        const { number } = this.state;  //state를 조회할 때는 this.state로 조회한다.
        return(
            <div>
                <h1>{ number }</h1>
                <button onClick={ ()=> { this.setState({ number: number + 1 }) } }> +1 </button>
            </div>
        );
    }
}

export default Counter;
```

```
import { useState } from "react";
const Counter = () => {
    const [number, setNumber] = useState(0);
    return(
        <div>
            <h1>{number}</h1>
            <button onClick={()=>setNumber(number+1)}> +1 </button>
        </div>
    )
}
export default Counter;
```

[src]-[App.js]

```
import React from 'react';
import Counter from './Counter';

const App = () => {
    return <Counter />
);

export default App;
```

- **state 객체 안에 여러 값이 있을 때**

[src]-[Counter.js]

```
import { Component } from "react";

class Counter extends Component{
    constructor(props){
        //현재 클래스형 컴포넌트가 상속받고 있는 생성자 함수를 호출
        super(props);
        //state의 초깃값 설정
        this.state = {
            number: 0,
            fixedNumber: 0
        };
    }

    render(){
        const { number, fixedNumber } = this.state;  //state를 조회할 때는 this.state로 조회한다.
        return(
            <div>
                <h1>{ number }</h1>
                <h2>바뀌지 않는 값: { fixedNumber }</h2>
                //this.setState를 사용하여 state에 새로운 값을 넣을 수 있다.
                <button onClick={ ()=> { this.setState({ number: number + 1 }) } }> +1 </button>
            </div>
        );
    }
}

export default Counter;
```

- **state를 constructor에서 꺼내기 : constructor 메서드를 선언하지 않고도 state 초깃값을 설정할 수 있다.**

[src]-[Counter.js]

```
import { Component } from "react";

class Counter extends Component{
    state = {
        number: 0,
        fixedNumber: 0
    };

    render(){
        const { number, fixedNumber } = this.state;  //state를 조회할 때는 this.state로 조회한다.
        return(
            ...
        );
    }
}

export default Counter;
```

- **this.setState가 끝난 후 특정 작업 실행하기 : setState를 사용하여 값을 업데이트하고 난 다음에 특정 작업을 하고 싶을 때는 setState의 두 번째 파라미터로 콜백함수를 등록해준다.**

[src]-[Counter.js]

```
<button onClick={() => {
    this.setState({
        number: number + 1
    },
    () => {
        console.log('방금 setState가 호출되었습니다.');
        console.log(this.state);
    });
}}>
    +1
</button>
```

- 함수형 컴포넌트에서 useState 사용하기 : useState 함수의 인자에는 상태의 초깃값을 넣어 준다. 클래스형 컴포넌트에서의 state 초깃값은 객체 형태를 넣어 주어야 하는데 useState에서는 반드시 객체가 아니어도 상관없다. 함수를 호출하면 배열이 반환되는데 배열의 첫 번째 원소는 현재 상태이고, 두 번째 원소는 상태를 바꾸어주는 함수이다. 이 함수를 세터(Setter) 함수라고 부른다. 그리고 배열 비구조화 할당을 통해 이름을 자유롭게 정해 줄 수 있다.

[src]-[Say.js]

```
import { useState } from "react"

const Say = () => {
    const [message, setMessage] = useState('');

    const onClickEnter = () => setMessage('안녕하세요!');
    const onClickLeave = () => setMessage('안녕히 가세요!');

    return(
        <div>
            <button onClick={onClickEnter}>입장</button>
            <button onClick={onClickLeave}>퇴장</button>
            <h1>{message}</h1>
        </div>
    );
}

export default Say;
```

[src]-[App.js]

```
import React from 'react';
import Counter from './Say';

const App = () => {
    return <Say />
);

export default App;
```

- 한 컴포넌트에서 useState 여러 번 사용하기 : useState는 한 컴포넌트에서 여러 번 사용해도 상관없다. 클래스형 컴포넌트나 함수형 컴포넌트 사용 시 주의할 점은 state 값을 바꾸어야 할 때 setState 혹은 useState를 통해 전달받은 세터 함수를 사용해야 한다.

[src]-[Say.js]

```
import { useState } from "react"

const Say = () => {
    const [message, setMessage] = useState('');
    const onClickEnter = () => setMessage('안녕하세요!');
    const onClickLeave = () => setMessage('안녕히 가세요!');

    //useState는 한 컴포넌트에서 여러 번 사용해도 상관없다.
    const [textColor, setTextColor] = useState('black');

    return(
        <div>
            <button onClick={onClickEnter}>입장</button>
            <button onClick={onClickLeave}>퇴장</button>
            <h1 style={{ color:textColor }}>{message}</h1>
            <button style={{ color: 'red' }}    onClick={() => setTextColor('red')}>빨간색</button>
            <button style={{ color: 'green' }} onClick={() => setTextColor('green')}>초록색</button>
            <button style={{ color: 'blue' }}   onClick={() => setTextColor('blue')}> 파란색</button>
        </div>
    );
}

export default Say;
```

# 4장 이벤트 핸들링

사용자가 브라우저에서 DOM 요소들과 상호 작용하는 것을 이벤트(event)라고 한다. 예를 들어 마우스 커서를 올려놓았을 때는 mouseover 이벤트를 실행하고 클릭했을 때는 onClick 이벤트를 실행한다. Form 요소는 값이 바뀔 때 onChange 이벤트를 실행한다. 이벤트를 사용할 때 주의 사항은 아래와 같다.

- 이벤트 이름은 카멜 표기법으로 작성한다.
- 이벤트에 실행할 자바스크립트의 코드를 전달하는 것이 아니라, 함수 형태의 값을 전달한다.
- DOM 요소에만 이벤트를 설정할 수 있다.

- onClick 이벤트 설정

[src]-[EventHandling.js]

```
import { Component} from "react";

class EventHandling extends Component {
    state = {
        isToggleOn: true
    }

    handleClick = () =>{
        this.setState({
            isToggleOn: !this.state.isToggleOn
        });
    }

    render(){
        return(
            <div>
                <button onClick={ this.handleClick }>
                    { this.state.isToggleOn ? 'ON' : 'OFF' }
                </button>
            </div>
        );
    }
}

export default EventHandling;
```

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

const EventHandling = () => {
    const [isToggleOn, setIsToggleOn] = useState(true);
    const handleClick = () => {
        setIsToggleOn(!isToggleOn)
    }
    return (
        <div>
            <button onClick={ handleClick }>
                { isToggleOn ? 'ON' : 'OFF' }
            </button>
        </div>
    );
};

export default EventHandling;
```

- onChange 이벤트 설정

[src]-[EventPractice.js]

```
class EventPractice extends Component {
    state = { message: '' }

    render(){
        return(
            <div>
                <h1>[이벤트 연습]</h1>
                <input type="text"
                    name="message"
                    placeholder="아무거나 입력 하세요!"
                    value={ this.state.message }
                    onChange={
                        (e)=>this.setState({message: e.target.value})}/>
                <button
                    onClick = { ()=> {
                        alert(this.state.message);
                        this.setState({ message: '' }); }}>
                    확인
                </button>
                <h1>{ this.state.message }</h1>
            </div>
        )
    }
}

export default EventPractice;
```

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

const EventPractice = () => {
    const [message, setMessage] = useState('');

    return (
        <div>
            <h1>[이벤트연습]</h1>
            <input type="text"
                name="message"
                placeholder="아무거나 입력 하세요!"
                value={message}
                onChange={(e)=>setMessage(e.target.value)}/>
            <button
                onClick= {() => {
                    alert(message);
                    setMessage('');} }>
                확인
            </button>
            <h1>{message}</h1>
        </div>
    );
};

export default EventPractice;
```

- **onKeyPress 이벤트 핸들링**

[src]-[EventPractice.js]

```
import { Component } from "react";

class EventPractice extends Component{
    state = {
        username: '',
        message: ''
    }

    //onChange 이벤트 핸들러에서 e.target.name은 input의 name을 가리킨다.
    handleChange = (e) => {
        this.setState({
            [e.target.name]: e.target.value
        });
    }

    handleClick = () => {
        alert(this.state.username + ':' + this.state.message);
        this.setState({
            username: '',
            message: ''
        });
    }

    //message input에서 Enter키를 눌렀을 때 handleClick 메서드를 호출한다.
    handleKeyPress = (e) => {
        if(e.key === 'Enter') {
            this.handleClick();
        }
    }

    render(){
        return(
            <div>
                <h1>[이벤트 연습]</h1>
                <input type="text"
                    name="username"
                    placeholder="사용자명"
                    value={this.state.username}
                    onChange={ this.handleChange } />
                <input
                    type="text"
                    name="message"
                    placeholder="아무거나 입력 하세요!"
                    value={ this.state.message }
                    onChange={ this.handleChange }
                    onKeyPress={ this.handleKeyPress } />
                <button onClick={ this.handleClick }>확인</button>
            </div>
        )
    }
}

export default EventPractice;
```

객체 안에서 key를 [ ]로 감싸면 그 안에 넣은 레퍼런스가 가리키는 실제 값이 key 값으로 사용된다. 예를 들어 아래와 같은 객체를 만들면

```
const name ='variantKey';
const object = {
    [name]: 'value'
};
```

결과는 다음과 같다.

```
{
    'variantKey': 'value'
}
```

[src]-[EventPractice.js]

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

const EventPractice = () => {
    const [username, setUsername] = useState('');
    const [message, setMessage] = useState('');

    const onChangeUsername = e => setUsername(e.target.value);
    const onChangeMessage = e => setMessage(e.target.value);
    const onClick = () => {
        alert(username + ':' + message);
        setUsername('');
        setMessage('');
    };
    const onKeyPress = e => {
        if(e.key === 'Enter') onClick();
    }

    return(
        <div>
            <h1>[이벤트 연습]</h1>
            <input type="text" name="username" placeholder="사용자명" value={username}
                onChange={ onChangeUsername }/>
            <input type="text" name="message" placeholder="아무거나 입력 하세요!" value={message}
                onChange={ onChangeMessage } onKeyPress={ onKeyPress }/>
            <button onClick={ onClick }>확인</button>
        </div>
    );
}
export default EventPractice;
```

**여러 개의 인풋 상태를 관리하기 위해 아래와 같이 e.target.name을 활용해서 useState에서 form 객체를 사용할 수 있다.**

[src]-[EventPractice.js]

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

const EventPractice = () => {
    const [form, setForm] = useState({
        username: '', message: ''
    });
    const { username, message } = form;
    const onChange = e => {
        const nextForm = {
            ...form,  //기존의 form 내용을 이 자리에 복사한 한다.
            [e.target.name]: e.target.value  //원하는 값을 덮어씌운다.
        };
        setForm(nextForm);
    };
    const onClick = () => {
        alert(username + ':' + message);
        setForm({ username: '', message: '' });
    };
    const onKeyPress = e => {
        if(e.key === 'Enter') onClick();
    }

    return(
        <div>
            <h1>이벤트 연습</h1>
            <input type="text" name="username" placeholder="사용자명" value={ username }
                onChange={ onChange }/>
            <input type="text" name="message" placeholder="아무거나 입력해보세요" value={ message }
                onChange={ onChange } onKeyPress={ onKeyPress }/>
            <button onClick={ onClick }>확인</button>
        </div>
    )
}

export default EventPractice;
```

- **ref란?**

HTML에서 id를 사용하여 DOM에 이름을 다는 것처럼 리액트 내부에서 DOM 이름은 ref(reference)를 사용한다. 리액트 컴포넌트 안에서도 id를 사용할 수 있지만 같은 컴포넌트를 여러 번 사용하는 경우 중복 id를 가진 DOM이 여러 개 생긴다. ref는 전역으로 작동하지 않고 컴포넌트 내부에서만 작동하기 때문에 이런 문제가 생기지 않는다. DOM에 직접적으로 접근해서 ref를 사용해야 하는 경우는 아래와 같다.

- 특정 input에 포커스 주기
- 스크롤 박스 조작하기
- Canvas 요소에 그림 그리기 등

- **콜백 함수를 통한 ref 설정**

ref를 만드는 가장 기본적인 방법은 콜백 함수를 사용하는 것이다. ref를 달고자 하는 요소에 ref라는 콜백 함수를 props로 전달해 주면 된다. 이 콜백 함수는 ref 값을 파라미터로 전달 받는다. 그리고 함수 내부에서 파라미터로 받은 ref를 컴포넌트의 멤버 변수로 설정해 준다.

[src]-[ValidationSample.css]

```
.success{
    background-color: lightgreen;
}

.failure{
    background-color: lightcoral;
}
```

input에서 onChange 이벤트가 발생하면 handleChange를 호출하여 state의 password 값을 업데이트 했다. button에서는 onClick 이벤트 발생시 handleButtonClick을 호출하여 clicked 참으로 설정했고, validated 값을 검증 결과로 설정했다. input의 className 값은 버튼을 누르기 전에는 비어있는 문자열을 전달하며 버튼을 누른 후에는 검증 결과에 따라 success 또는 failure값을 설정한다.

[src]-[ValidationSample.js]

```
import { Component } from "react";
import './ValidationSample.css';

class ValidationSample extends Component{
    state = {
        password: '',
        validated: false,
        clicked: false
    }

    handleChange = (e) => {
        this.setState({
           password: e.target.value,
        });
    }

    handleButtonClick = () => {
        this.setState({
            clicked: true,
            validated: this.state.password === '0000'
        });
        this.input.focus();
    }

    render(){
        return(
            <div>
                <input type="password"
                    value={this.state.password}
                    onChange={ this.handleChange }
                    className={ this.state.clicked ? (this.state.validated ? 'success': 'failure') : '' }
                    ref={ (ref)=>{ this.input=ref } }/>
                <button onClick={ this.handleButtonClick }>검증하기</button>
            </div>
        )
    }
}

export default ValidationSample;
```

[src]-[ValidationSample.js]

```
import React, { useState, useRef } from 'react';
import './ValidationSample.css';

const ValidationSample = () => {
    const [password, setPassword] = useState('');
    const [validate, setValidate] = useState(false);
    const [clicked, setClicked] = useState(false);

    const input1 = useRef(null);

    const onClick = () => {
        setClicked(true);
        setValidate(password === '0000');
        input1.current.focus();
    }

    const onChange = (e) => {
        setPassword(e.target.value);
    }

    return (
        <div>
            <input type="password"
                value={password}
                onChange={onChange}
                className={clicked ? (validate ? 'success':'failure') : ''}
                ref={input1}/>
            <button onClick={onClick}>검증하기</button>
        </div>
    );
};

export default ValidationSample;
```

- createRef를 통한 ref 설정

ref를 만드는 또 다른 방법은 리액트에 내장되어 있는 createRef라는 함수를 사용하는 것이다. 이 함수를 사용해서 만들면 더 적은 코드로 쉽게 사용할 수 있다. 이 기능은 리액트 v16.3부터 도입되었으며 이전 버전에서는 작동되지 않는다.

createRef를 사용하여 ref를 만들려면 우선 컴포넌트 내부에서 멤버 변수로 React.createRef()를 담아 주어야 한다. 그리고 해당 멤버 변수를 ref를 달고자 하는 요소에 ref props로 넣어 주면 ref 설정이 완료된다. 설정 후 ref를 설정해준 DOM에 접근하려면 this.input.current를 조회하면 된다. 앞으로 두 가지 방법 중에서 편한 방법을 사용하면 된다.

[src]-[RefSample.js]

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

class RefSample extends Component {
    input = React.createRef();

    handleFocus = () => {
        this.input.current.focus();
    }

    render() {
        return(
            <div>
                <input ref={ this.input }/>
            </div>
        )
    }
}

export default RefSample;
```

• 컴포넌트에 ref 달고 내부 메서드 사용하기

리액트에서는 컴포넌트에도 ref를 달 수 있다. 이 방법은 주로 컴포넌트 내부에 있는 DOM을 컴포넌트 외부에서 사용할 때 쓴다. 컴포넌트에 ref를 다는 방법은 DOM에 ref를 다는 방법과 똑같다.

아래는 스크롤 박스가 있는 컴포넌트를 하나 만들고 스크롤바를 아래로 내리는 작업을 부모 컴포넌트에서 실행하는 예제이다. 프로그램을 작성하는 순서는 아래와 같다.

1) ScrollBox 컴포넌트 만들기
2) 컴포넌트 ref달기
3) ref를 이용하여 컴포넌트 내부 메서드 호출하기

자바스크립트로 스크롤바를 내릴 때는 DOM 노드가 가진 다음 값들을 사용한다.

- scrollTop: 새로 스크롤바의 위치 (0 ~ 350)
- scrollHeight: 스크롤이 있는 박스 안의 div 높이 (650)
- clientHeight: 스크롤이 있는 박스의 div 높이 (300)

[src]-[ScrollBox.js]

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

class ScrollBox extends Component{
    scrollToBottom = () => {
        const {scrollHeight, clientHeight} = this.box;  //scrollHeight=this.box.scrollHeight; s=clientHeight=this.box.clientHeight;
        this.box.scrollTop = scrollHeight - clientHeight;
    }

    render(){
        const style = {
            border: '1px solid black',
            height: '300px',
            width: '300px',
            overflow: 'auto',
            position: 'relative'
        };

        const innerStyle = {
            width: '100%',
            height: '650px',
            background: 'linear-gradient(white, black)'
        }

        return(
            <div style={ style } ref={ (ref)=>{ this.box=ref } }>  //최상위 DOM에 ref를 달아 준다.
                <div style={ innerStyle }></div>
            </div>
        );
    }
}
export default ScrollBox;
```

App 컴포넌트에서 ScrollBox에 ref를 달고 버튼을 만들어 누르면 ScrollBox 컴포넌트의 ScrollToBottom 메서드를 실행하도록 작성한다.

[src]-[App.js]

```
import { Component } from 'react';
import ScrollBox from './ScrollBox';

class App extends Component{
    render(){
        return(
            <div>
               <ScrollBox ref={ (ref)=>{ this.scrollBox = ref } }/>
               <button onClick={ ()=> this.scrollBox.scrollToBottom() }>맨 밑으로</button>
            </div>
        );
    }
}

export default App;
```

- **데이터 배열을 컴포넌트 배열로 변환 : 자바스크립트 배열 객체의 내장 함수인 map 함수를 사용해 반복되는 컴포넌트를 렌더링 한다.**

[src]-[IterationSample.js]

```
const IterationSample = () => {
    const names = [ '눈사람', '얼음', '눈', '바람' ];
    const namesList = names.map(name=><li>{name}</li>);
    //console창에 "key" prop이 없다는 경고 메시지가 출력된다.
    //const namesList = names.map((name,index)=><li key={index}>{name}</li>);
    return(
        <div><ol>{namesList}</ol></div>
    )
}

export default IterationSample;
```

- **데이터 추가 기능 구현 : 배열의 내장 함수 concat을 사용하여 항목을 추가한 배열을 만들고 setNames를 통해 상태를 업데이트해준다.**

[src]-[IterationSample.js]

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

const IterationSample = () => {
    const [names, setNames] = useState([
        {id:1, text: '눈사람'}, {id:2, text: '얼음'}, {id:3, text: '눈'}, {id:4, text: '바람'}
    ]);
    const [inputText, setInputText] = useState('');
    const [nextId, setNextId] = useState(5);  //새로운 항목을 추가할 때 사용할 id

    const onChange = e => setInputText(e.target.value);
    const onClick = () => {
        const nextNames = names.concat({  //push는 기존 배열 자체를 변경해 주는 반면 concat은 새로운 배열을 만들어 준다. (불변성유지)
            id: nextId,
            text: inputText
        });
        setNextId(nextId + 1);  //nextId 값에 1을 더해준다.
        setNames(nextNames);  //names 값을 업데이트한다.
        setInputText('');  //inputText를 비운다.
    };

    const namesList=names.map(name => <li key={ name.id }>{ name.text }</li>);

    return(
        <div>
            <input value={ inputText } onChange={ onChange }/>
            <button onClick={ onClick }>추가</button>
            <ol>{ namesList }</ol>
        </div>
    )
}
export default IterationSample;
```

- **데이터 제거 기능 구현하기 : 불변성을 유지하면서 배열의 특정 항목을 지울 때는 배열의 내장 함수 filter를 사용한다.**

[src]-[IterationSample.js]

```
const IterationSample = () => {
    ...
    const onRemove = id => {
        const nextNames = names.filter(name => name.id !== id);  //filter 함수를 사용해 배열에서 특정조건을 만족하는 원소들만 분류한다.
        setNames(nextNames);
    };

    const namesList=names.map(name => (
        <li key={ name.id } onDoubleClick={ ()=>onRemove(name.id) }>
            { name.text }
        </li>
    ));
    return(...)
}
```

# 5장 컴포넌트의 라이프사이클 메서드

- 마운트: DOM이 생성되고 웹 브라우저상에 나타나는 것을 마운트라고 한다. 이때 호출되는 메서드는 다음과 같다.

1) constructor: 컴포넌트를 새로 만들 때마다 호출되는 클래스 생성자 메서드이다.
2) render: 우리가 준비한 UI를 렌더링하는 메서드이다.
3) componentDidMount: 컴포넌트가 웹 브라우저상에 나타난 후 호출하는 메서드이다. (처음 렌더링될 때 호출)

- 업데이트: pops가 바뀔 때, state가 바뀔 때, 부모 컴포넌트가 리렌더링될 때 다음 메서드를 호출한다.

1) shouldComponentUpdate: props 또는 state를 변경했을 때 컴포넌트가 리렌더링을 해야 할지 말아야 할지를 결정하는 메서드이다.
2) render: 컴포넌트를 리렌더링 한다.
3) componentDidUpdate: 컴포넌트의 업데이트 작업이 끝난 후 호출하는 메서드이다.

- 언마운트: 마운트의 반대 과정, 즉 컴포넌트를 DOM에서 제거하는 것을 언마운트라고 한다.

componentWillUnmount: 컴포넌트가 웹 브라우저상에서 사라지기 전에 호출하는 메서드이다.

- 라이프사이클 메서드 사용하기

[src]-[LifeCycleSample.js]

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

class LifeCycleSample extends Component{
    state = { number: 0 };

    constructor(props) {
        super(props);
        console.log('1.constructor...');
    }

    //다른 자바스크립트 라이브러리 또는 프레임워크의 함수를 호출하거나 이벤트등록, 네트워크 요청 같은 비동기 작업처리
    componentDidMount() {
        console.log('3.componentDidMount...');
    }

    //새로 설정될 props 또는 state를 nextProps와 nextState로 접근
    shouldComponentUpdate(nextProps, nextState) {
        console.log('2-0,shouldComponentUpdate...', nextProps, nextState);
        return nextState.number % 3 !== 0;  //숫자의 마지막 자리가 3이면 리렌더링하지 않는다.
    }

    //이전 props 또 state를 prevProps 또는 prevState로 접근
    componentDidUpdate(preProps, prevState) {
        console.log('2-2,componentDidUpdate...', preProps, prevState);
    }

    //componentDidMount에서 등록한 이벤트, 타이머 등을 여기서 제거한다.
    componentWillUnmount() {
        console.log('4.componentWillUnmount...');
    }

    handleClick = () => {
        this.setState({
            number: this.state.number + 1
        });
    }

    render(){
        console.log('2-1.render...');
        return(
            <div>
                <h1 style={{ color:this.props.color }}>{ this.state.number }</h1>
                <button onClick={ this.handleClick }>더하기</button>
            </div>
        );
    }
}

export default LifeCycleSample;
```

[src]-[App.js]

```
import { Component } from 'react';
import LifeCycleSample from './LifeCycleSample';

class App extends Component{
    state = {
        color: 'red'
    };
    handleClick = () => {
        this.setState({
            color: this.state.color === 'red' ? 'blue' : 'red'
        });
    }

    render(){
        return(
            <div>
                <button onClick={ this.handleClick } style={{ color:this.state.color }}>
                    { this.state.color }
                </button>
                {/* LifeSample 컴포넌트에 color 값을 props로 설정* /}
                <LifeCycleSample color={ this.state.color }/>
            </div>
        );
    }
}

export default App;
```

- **API 호출 예제**

JSON Placeholder 사이트: https://jsonplaceholder.typicode.com

[src]-[ApiExample.js]

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

class ApiExample extends Component{
    constructor(props){
        super(props);
        this.state = {
            data: ''
        };
    }

    callAPI = () => {
        fetch('https://jsonplaceholder.typicode.com/todos/2')
        .then(res => res.json())
        .then(json => {
            this.setState({
                data: json.title
            });
        });
    }

    componentDidMount(){
        this.callAPI();
    }

    render(){
        return(
            <div>
                <h3>Title:{ this.state.data ? this.state.data : '데이터를 불러오는 중입니다.' }</h3>
            </div>
        );
    }
}

export default ApiExample;
```

```
const ApiExample = () => {  //함수형 컴포넌트
    const [data, setData] = useState('')
    const [number, setNumber] = useState(1);

    const callAPI = ()=> {
        fetch(`.../${number}`)
        .then(res => res.json())
        .then(json => {
            setData(json.title);
        });
    }

    useEffect(() => {
        callAPI();
    }, [number]);

    return(
       ...
    )
}

const [todos, setTodos] = useState([]);
setTodos(json.filter(todo => todo.id <= 5));
```

# 6장 Hooks

Hooks는 리액트 v16.8에 도입된 기능으로 함수형 컴포넌트에서도 상태 관리를 할 수 있는 useState, 렌더링 직후 작업을 설정하는 useEffect 등의 기능을 제공하여 기존의 함수형 컴포넌트에서 할 수 없었던 다양한 직업을 할 수 있게 해 준다.

## • useState

useState는 가장 기본적인 Hook이며 함수형 컴포넌트에서 가변적인 상태를 지닐 수 있도록 해준다.

### • useState 기능을 사용해서 숫자 카운터 구현

[src]-[Counter.js]

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

const Counter = () => {
    const [value, setValue] = useState(0);
    return(
        <div>
            <h2>현재 카운터 값은 <b>{value}</b>입니다.</h2>
            <button onClick={ ()=> setValue(value - 1) }>1감소</button>
            <button onClick={ ()=> setValue(value + 1) }>1증가</button>
        </div>
    );
};

export default Counter;
```

```
const [value, setValue] = useState(0);

const onClickAdd = () =>{
    setValue(value+1);
}
const onClickReduce = () => {
    setValue(value-1);
}

return(
    <div>
        <h2>현재 카운트 값은 <b>{value}</b>입니다.</h2>
        <button onClick={ onClickReduce }>1감소</button>
        <button onClick={ onClickAdd }>1증가</button>
    </div>
)
```

useState는 코드 상단에서 import 구문을 통해 불러오고 아래와 같이 사용한다.

```
const [value, setValue] = useState(0);
```

useState 함수의 파라미터에는 상태의 기본값을 넣어준다. 이 함수가 호출되면 배열이 반환된다. 그 배열의 첫 번째는 상태 값, 두 번째는 상태를 설정하는 함수이다. 이 함수에 파라미터를 넣어서 호출하면 전달받은 파라미터로 값이 바뀌고 컴포넌트가 정상적으로 리렌더링된다.

### • useState를 여러 번 사용하기

하나의 useState 함수는 하나의 상태 값만 관리할 수 있다. 컴포넌트에서 관리해야 할 상태가 여러 개라면 useState를 여러 번 사용한다.

[src]-[Info.js]

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

const Info = () => {
    const [name, setName] = useState('');
    const [nickname, setNickname] = useState('');
    const onChangeName = e => {
        setName(e.target.value);
    };
    const onChangeNickname = e => {
        setNickname(e.target.value);
    };

    return(
        <div>
            <div>
                <input value={ name } onChange={ onChangeName }/>
                <input value={ nickname } onChange={ onChangeNickname }/>
            </div>
            <div>
                <h1>이름:{ name }</h1>
                <h1>닉네임:{ nickname }</h1>
            </div>
        </div>
    );
};

export default Info;
```

- **useEffect**

**useEffect는 컴포넌트가 렌더링될 때마다 특정 작업을 수행하도록 설정할 수 있는 Hook이다. 클래스형 컴포넌트의 componentDidMount와 componentDidUpdate를 합친 형태로 보아도 무방하다.**

- **Info 컴포넌트에 useEffect를 적용**

**[src]-[Info.js]**

```
import { useEffect, useState } from "react";

const Info = () => {
    ...
    useEffect(() => {
        console.log('렌더링이 완료되었습니다!');
        console.log({ name, nickname });
    });
    return( ... );
};

export default Info;
```

- **마운트될 때만 실행하고 싶을 때**

**useEffect에서 설정한 함수를 컴포넌트가 화면에 처음 렌더링 될 때만 실행하고 업데이트될 때는 실행하지 않으려면 함수의 두 번째 파라미터로 비어 있는 배열을 넣어 준다. 코드를 실행하면 컴포넌트가 처음 나타날 때만 콘솔에 문구가 나타나고 그 이후에는 나타나지 않을 것이다.**

**[src]-[Info.js]**

```
import { useEffect, useState } from "react";

const Info = () => {
    ...
    useEffect(() => {
        console.log('마운트될 때만 실행됩니다.');
    }, []);  //두 번째 파라미터를 Dependency Array라 한다.
     return( ... );
};

export default Info;
```

- **특정 값이 업데이트될 때만 실행하고 싶을 때 클래스 컴포넌트를 경우**

**[src]-[Info.js]** 클래스형 컴포넌트라면 다음과 같이 작성한다.

```
componentDidUpdate(prevProps, prevState) {
    //이 코드는 props 안에 들어 있는 value 값이 바뀔 때만 특정 작업을 수행한다.
    if(prevProps.value != this.props.value) {
        doSomething();
    }
}
```

**useEffect에서의 사용법은 아래와 같다. useEffect의 두 번째 파라미터로 전달되는 배열 안에 검사하고 싶은 값을 넣어주면 된다. 배열 안에는 useState를 통해 관리하고 있는 상태를 넣어 주어도 되고 props로 전달 받은 값을 넣어줘도 된다.**

**[src]-[Info.js]**

```
import { useEffect, useState } from "react";

const Info = () => {
    ...
    useEffect(() => {
        console.log('name값 업데이트:' + name);
    }, [name]);
     return( ... );
};

export default Info;
```

- **뒷정리하기**

useEffect는 기본적으로 렌더링되고 난 직후마다 실행되며, 두 번째 파라미터 배열에 무엇을 넣는지에 따라 실행되는 조건이 달라진다. 컴포넌트가 언마운트되기 전이나 업데이트되기 직전에 어떠한 작업을 수행하고 싶다면 useEffect에서 뒷정리(cleanup)함수를 반환해 준다.

[src]-[Info.js]

```
useEffect(() => {
    console.log('effect...' + name);
    //뒷정리 함수
    return ()=> {
        console.log('cleanup...' + name);
    }
});  //렌더링될 때마다 뒷정리 함수가 호출 되어 업데이트되기 직전의 값을 보여준다.
```

아래 예제는 언마운트될 때와 name 값이 업데이트될 때 뒷정리 함수가 계속 나타나는 것을 확인할 수 있다. 그리고 뒷정리 함수가 호출될 때는 업데이트되기 직전의 값을 보여준다. 오직 언마운트될 때만 뒷정리 함수를 호출하고 싶다면 useEffect 함수의 두 번째 파라미터에 비어있는 배열을 넣으면 된다.

[src]-[Info.js]

```
useEffect(() => {
    console.log('effect...' + name);
    return ()=> {  //뒷정리 함수
        console.log('cleanup...' + name);
    }
}, [name]);  //오직 언마운트될 때만 뒷정리 함수를 호출하고 싶다면 useEffect 함수의 두 번째 파라미터에 비어있는 배열을 넣으면 된다.
```

- **뒷정리 함수 예제**

[src]-[Timer.js]

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

const Timer = () => {
    useEffect(() => {
        const timer = setInterval(() => {
            console.log('타이머 돌아가는중...')
        }, 1000);
        return () => {
            clearInterval(timer);
            console.log('타이미가 종료되었습니다.')
        }
    },[])

    return (
        <div>
            <span>타이머를 시작합니다. 콘솔을 보세요!</span>
        </div>
    );
};
export default Timer;
```

[src]-[App.js]

```
import { useState } from 'react';
import './App.css';
import Timer from './Timer';

function App() {
    const [showTimer, setShowTimer] = useState(false);
    return(
        <div>
            { showTimer && <Timer/> }
            <button onClick={ () => setShowTimer(!showTimer) }>Toggle Timer</button>
        </div>
    )
}
export default App;
```

- API 호출 예제

[src]-[Info.js]

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

const APITodos = () => {
    const [todos, setTodos] = useState([]);
    const [page, setPage] = useState(1);

    const callAPI = () => {
        fetch(`https://jsonplaceholder.typicode.com/todos`)
        .then(res => res.json())
        .then(json => {
            setTodos(json.filter( todo =>
                todo.id >= (page-1)*5+1 && todo.id <= page*5)
            );
        });
    }

    useEffect(()=>{
        callAPI();
    });

    return (
        <div>
            { todos ? todos.map(todo => <h3 key={todo.id}>[{todo.id}] {todo.title}</h3>) : '데이터를 불러오는 중...' }
            <button onClick={()=>{setPage(page-1)}}>이전</button> 
            <button onClick={()=>{setPage(page+1)}}>다음</button>
        </div>
    );
};
export default APITodos;
```

```
...
fetch(`https://jsonplaceholder.typicode.com/todos/${number}`)
    .then(res => res.json())
    .then(json => {
        setData(json.title);
    });
}
...
return (
    <div>
        <h3>{ data ? data : '데이터를 불러오는 중입니다...' }</h3>
    </div>
);
...
```

- useReducer

useReducer는 useState보다 더 다양한 컴포넌트 상황에 따라 다양한 상태를 다른 값으로 업데이트해 주고 싶을 때 사용하는 Hook이다. 리듀서는 현재 상태, 그리고 업데이트를 위해 필요한 정보를 담은 액션(action)값을 전달받아 새로운 상태를 변환하는 함수이다.

- 카운트 구현하기

useReducer의 첫 번째 파라미터에는 리듀서 함수를 넣고, 두 번째 파라미터에는 리듀서의 기본값을 넣어 준다. 이 Hook을 사용하면 state와 dispatch 함수를 받아온다. 여기서 state는 현재 가리키고 있는 상태이고, dispatch는 액션을 발생시키는 함수이다. dispatch(action)과 같은 형태로, 함수 안에 파라미터로 액션 값을 넣어 주면 리듀서 함수가 호출되는 구조이다. useReducer를 사용했을 때의 가장 큰 장점은 컴포넌트 업데이트 로직을 컴포넌트 바깥으로 빼낼 수 있다는 것이다.

[src]-[Counter.js]

```
import { useReducer } from "react";
function reducer(state, action){  //action, type에 따라 다른 작업 수행
    switch(action.type){
        case 'INCREMENT':
            return { value: state.value + 1 };
        case 'DECREMENT':
            return { value: state.value + 1 };
        default:
            return state;  //아무것도 해당되지 않을 때 기존 상태 변환
    }
}

const Counter = () => {
    const [state, dispatch] = useReducer(reducer, { value: 0 });
    return(
        <div>
            <h2>현재 카운터 값은 <b>{state.value}</b>입니다.</h2>
            <button onClick={ ()=> dispatch({ type: 'INCREMENT' }) }>+1</button>
            <button onClick={ ()=> dispatch({ type: 'DECREMENT' }) }>-1</button>
        </div>
    );
};
export default Counter;
```

- **Input 상태 관리하기**

기존에는 Input이 여러 개여서 useState를 여러 번 사용했으나 useReducer를 사용하면 기존에 클래스형 컴포넌트에서 input 태그에 name값을 할당하고 e.target.name을 참조하여 setState를 해준 것과 유사한 방식으로 작업을 처리할 수 있다.

[src]-[Info.js]

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

function reducer(state, action){
    return { ...state, [action.name]: action.value  };
}

const Info = () => {
    const [state, dispatch] = useReducer(
        reducer, { name: '', nickname: '' }
    );
    const { name, nickname } = state;
    const onChange = e => { dispatch(e.target); };

    return(
        <div>
            <div>
                <input name="name" value={ name } onChange={ onChange }/>
                <input name="nickname" value={ nickname } onChange={ onChange }/>
            </div>
            <div>
                <h1>이름: { name }</h1>
                <h1>닉네임: { nickname }</h1>
            </div>
        </div>
    );
};
export default Info;
```

- **useMemo**

useMemo(Memoization)를 사용하면 함수형 컴포넌트 내부에서 발생하는 연산을 최적화할 수 있다. 렌더링하는 과정에서 특정 값이 바뀌었을 때만 연산을 실행하고 원하는 값이 바뀌지 않았다면 이전에 연산했던 결과를 다시 사용하는 방식이다.

- **useMemo를 이용한 계산기 컴포넌트**

[src]-[Calculater.js]

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

const wait = (sec) => {
    let start=Date.now();
    let now = start;
    while(now-start < sec * 1000){
        now = Date.now();
    }
}

const hardCalculate = (number)=>{
    wait(2);
    for(let i=0; i < 9999999; i++) {}
    console.log('어려운 계산');
    return number + 10000;
};

const easyCalculate = (number)=>{
    console.log('쉬운 계산');
    return number + 1;
};

const Calculator = () => {
    const [hardNumber, setHardNumber] = useState(1);
    const [easyNumber, setEasyNumber] = useState(1);

    //const hardSum = hardCalculate(hardNumber);
```

```
    const hardSum = useMemo(() => {
        return hardCalculate(hardNumber);
    }, [hardNumber]);

    const easySum = easyCalculate(easyNumber);

    return (
        <div>
            <h3>어려운 계산기</h3>
            <input
                type="number"
                value={ hardNumber }
                onChange={ (e)=> setHardNumber(parseInt(e.target.value))}/>
            <span> + 10000 = { hardSum } </span>

             <h3>쉬운 계산기</h3>
             <input
                 type="number"
                 value={ easyNumber }
                 onChange={ (e)=> setEasyNumber(parseInt(e.target.value))}/>
              <span> + 1 = { easySum } </span>
        </div>
    );
};

export default Calculator;
```

- **useMemo 예제**

[src]-[Info.js]

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

const UseMemoExample = () => {
    const [number, setNumber] = useState(0);
    const [isKorea, setIsKorea] = useState(true);

    //원시 (Primitive) 타입
    //const location = isKorea ? '한국' : '외국';

    //객체 (Object) 타입
    //const location = {
        country: isKorea ? '한국' : '외국',
    }

    const location = useMemo(() => {
        return {
            country: isKorea ? '한국' : '외국',
        }
    }, [isKorea]);

    useEffect(() => {
        console.log('useEffect 호출')
    }, [isKorea]);

    return (
        <div>
            <h2>하루에 몇 끼 먹어요?</h2>
            <input
                type="number"
                value={number}
                onChange={ (e) => setNumber(e.target.value) }/>
            <hr/>
            <h2>어느 나라에 있어요?</h2>
            <p>나라: { location.country }</p>
            <button onClick={ () => setIsKorea(!isKorea) }>비행기 타자</button>
        </div>
    );
};

export default UseMemoExample;
```

- **useCallback**

**컴포넌트는 리렌더링 될 때마다 새로 만들어진 함수를 사용한다. 컴포넌트의 렌더링이 자주 발생하거나 렌더링해야 할 컴포넌트의 개수가 많아지면 이 부분을 최적화해 주는 것이 좋다. useCallback의 첫 번째 파라미터는 생성하고 싶은 함수를 넣고, 두 번째 파라미터에는 배열을 넣는다. 이 배열에는 어떤 값이 바뀌었을 때 함수를 새로 생성해야 하는지 명시해야 한다.**

[src]-[UseCallbackExample.js]

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


const UseCallbackExample = () => {
    const [number, setNumber] = useState(0);
    const [toggle, setToogle] = useState(true);
    /*
    const someFunction = () => {
        console.log(`someFunc: number: ${ number }`)
    }
    */

    const someFunction = useCallback(() => {
        console.log(`someFunc: number: ${ number }`)
        return;
    },[number]);

    useEffect(() => {
        console.log('someFunction이 변경되었습니다.');
    }, [someFunction]);

    return (
        <div>
            <input
                type="number"
                value={ number }
                onChange={ (e) => setNumber(e.target.value) }/>
            <br/>
            <button onClick={ someFunction }>Call someFucnction</button>
        </div>
    );
};

export default UseCallbackExample;
```

[src]-[UseCallbackExample.js]

```
const UseCallExample = () => {
    const [number, setNumber] = useState(0);
    const [toggle, setToogle] = useState(true);

    const someFunction = useCallback(() => {
        console.log(`someFunc: number: ${ number }`)
        return;
    },[number]);

    useEffect(() => {
        console.log('someFunction이 변경되었습니다.');
    }, [someFunction]);

    return (
        <div>
            <input
                type="number"
                value={ number }
                onChange={ (e) => setNumber(e.target.value) }/>
            <button onClick={ () => setToogle(!toggle) }>{ toggle.toString() }</button>  //값이 바뀌어도 useEffect가 실행되지 않는다.
            <br/>
            <button onClick={ someFunction }>Call someFucnction</button>
        </div>
    );
};

export default UseCallExample;
```

- **useCallback를 이용한 박스 사이즈변경 예제**

[src]-[Box.js]

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

const Box = ({ createBoxStyle }) => {
    const [style, setStyle] = useState({});

    useEffect(() => {
        console.log('박스 키우기');
        setStyle(createBoxStyle);
    }, [createBoxStyle]);

    return (
        <div style={ style }></div>
    );
};

export default Box;
```

[src]-[BoxSize.js]

```
import React, { useState } from 'react';
import Box from './Box';

const BoxSize = () => {
    const [size, setSize] = useState(100)
    const createBoxStyle = () => {
        return {
            backgroundColor: 'pink',
            width: `${size}px`,
            height: `${size}px`,
        };
    };

    return (
        <div>
            <input
                type="number"
                value={size}
                onChange={ (e) => setSize(e.target.value) }/>
            <Box createBoxStyle={createBoxStyle}/>
        </div>
    );
};

export default BoxSize;
```

- **위 예제에 배경색 변경기능 추가**

[src]-[Box.js]

```
const BoxSize = () => {
    const [size, setSize] = useState(100)
    const [isDark, setIsDark] = useState(false);

    const createBoxStyle = useCallback(() => {
        ...
    }, [size]);

    return (
        <div style={{ background: isDark ? 'black' : 'white' }}>
            <input
                .../>
            <button onClick={ ()=> setIsDark(!isDark) }>Change Theme</button>
            <Box createBoxStyle={createBoxStyle}/>
        </div>
    );
};
export default BoxSize;
```

- **useRef**

useRef Hook은 함수형 컴포넌트에서 ref를 쉽게 사용할 수 있다. 컴포넌트에서 등록 버튼을 눌렀을 때 포커스가 input 쪽으로 넘어가도록 아래와 같이 작성해 보자 useRef를 사용하여 ref를 설정하면 useRef를 통해 만든 객체 안의 current 값이 실제 엘리먼트를 가리킨다.

- useRef를 이용한 포커스 이동

[src]-[Average.js]

```
import { useCallback, useMemo, useRef, useState } from "react";

const getAverage = numbers => {
    if(numbers.length === 0) return 0;
    const sum = numbers.reduce((a, b) => a + b);
    return sum / numbers.length;
};
const Average = () => {
    const inputE1 = useRef(null);
    ...
    const onInsert = useCallback(e => {
        const nextList = list.concat(parseInt(number));
        setList(nextList);
        setNumber('');
        inputE1.current.focus();
    },[number, list]);  //number혹은 list가 바뀌었을 때만 함수 생성

    const avg = useMemo(() => getAverage(list), [list]);

    return(
        <div>
            <input value={ number } onChange={ onChange } ref={ inputE1 }/>
            <button onClick={ onInsert }>등록</button>
            <ul>
                {list.map((value, index)=> (<li key={ index }>{ value }</li>))}
            </ul>
            <h1>평균값:{ avg }</h1>
        </div>
    );
};

export default Average;
```

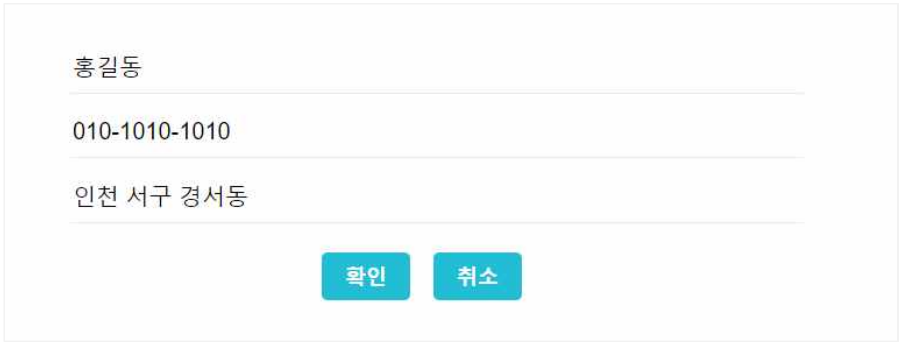
- 로컬 변수 사용하기

로컬 변수를 사용해야 할 때도 useRef를 활용할 수 있다. 여기서 로컬 변수란 렌더링과 상관없이 바뀔 수 있는 값을 의미한다. 아래는 ref 안의 값이 바뀌어도 컴포넌트가 렌더링되지 않는 예제이다. 렌더링과 관련되지 않는 값을 관리할 때만 이러한 방식으로 코드를 작성한다.

[src]-[RefSample.js]

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

const RefSample = () => {
    const id = useRef(1);

    const setId = (n) => {
        id.current = n;
    };
    const printId = () => {
        setId(100);
        console.log(id.current);  //ref안의 값이 바뀌어도 컴포넌트가 렌더링되지 않는다.
    };

    useEffect(() => {
        console.log('렌터링....');
    });

    return(
        <div onClick={ ()=>printId()} >refSample</div>
    );
};

export default RefSample;
```

- **Hooks 연습**




[src]-[Address.css]

```
.AddressForm {
    width: 500px;
    padding: 20px;
    margin: 0px auto;
}

.AddressForm  input {
    width: 100%;
    font-size: 1rem;
    border: none;
    padding-bottom: 0.5rem;
    outline: none;
    border-bottom: 1px solid #e2e2e2;
    margin-top: 1rem;
}

.AddressForm input:focus {
    border-bottom: 1px solid #495057;
}

.skyButton {
    border: none;
    border-radius: 4px;
    font-size: 0.9rem;
    font-weight: bold;
    padding: 0.4rem 1rem;
    color: white;
    background: #22b8cf;
    cursor: pointer;
    margin-right: 1rem;
}

.skyButton:hover {
    background: #3bc9db;
}

.buttons {
    text-align: center;
    margin: 20px;
}
```

[src]-[Address.js]

```
import React, { useCallback, useRef, useState, useEffect } from 'react';
import './Address.css';

const Address = () => {
    const [form, setForm] = useState({
        name: '홍길동',
        tel: '010-1010-1010',
        address: '인천 서구 경서동'
    });

    //비구조 할당
    const {name, tel, address} = form;

    const inputName = useRef('');

    useEffect(()=>{
        console.log('Renderning...');
    },[]);  //두 번째 파라미터에 빈 배열을 추가하면 처음 렌더링될 때만 실행하고 업데이트될 때는 실행하지 않는다.

    const onChange = useCallback(e=>{
        setForm({
            ...form,
            [e.target.name]: e.target.value
        })
        console.log('onChange....' + name);
    },[]);  //두 번째 파라미터에 빈 배열을 추가하면 컴포넌트가 처음 렌더링될 때만 함수 생성

    const onSubmit = (e) => {
        e.preventDefault();
        alert(`이름:${name}\n전화:${tel}\n주소:${address}`);
        onReset();
        inputName.current.focus();  useRef를 통해 만든 객체 안의 current 값이 실제 엘리먼트를 가리킨다.
    }

    const onReset = () => {
        setForm({
            name: '',
            tel: '',
            address:''
        });
    }

    return (
        <form className='AddressForm' onSubmit={ onSubmit }>
            <input name="name" placeholder="이름"
                value={name}
                onChange={ onChange }
                ref={ inputName }/>
            <input name="tel" placeholder="전화"
                value={tel}
                onChange={ onChange }/>
            <input name="address" placeholder="주소"
                value={address}
                onChange={ onChange }/>
            <div className='buttons'>
                <button type="submit" className="skyButton">확인</button>
                <button type="reset" onClick={ onReset } className="skyButton">취소</button>
            </div>
        </form>
    );
};

export default Address;
```

- **커스텀 Hooks 만들기**

여러 컴포넌트에서 비슷한 기능을 공유할 경우, 이를 여러분만의 Hook으로 작성하여 로직을 재사용할 수 있다.

- **커스텀 Hooks**

Info 컴포넌트에서 여러 개의 input을 관리하기 위해 useReducer로 작성했던 로직을 useInputs라는 Hook으로 따로 분리하여 작성한다.

[src]-[useInputs.js]

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

function reducer(state, action){
    return {
        ...state,
        [action.name]: action.value
    };
}

export default function useInputs(initialForm){
    const [state, dispatch] = useReducer(reducer, initialForm);
    const onChange = e => {
        dispatch(e.target);
    };

    return [state, onChange];
}
```

- **useInput Hooks를 info 컴포넌트에 사용**

[src]-[Info.js]

```
import useInputs from './useInputs';

const Info = () => {
    const [state, onChange] = useInputs(
        {
            name: '',
            nickname: ''
        });
   1.
    const { name, nickname } = state;

    return(
        <div>
            <div>
                <input name="name" value={ name } onChange={ onChange }/>
                <input name="nickname" value={ nickname } onChange={ onChange }/>
            </div>
            <div>
                <h1>이름: { name }</h1>
                <h1>닉네임: { nickname }</h1>
            </div>
        </div>
    );
};
```

- **다른 hooks**

이번에 커스텀 Hooks를 만들어서 사용했던 것처럼, 다른 개발자가 만든 Hooks도 라이브러리로 설치하여 사용할 수 있다. 다른 개발자가 만든 다양한 Hooks 리스는 아래 링크에서 확인할 수 있다.

```
- https://nikgraf.github.io/react-hooks/
- https://github.com/rehooks/awesome-react-hooks
```

- **정리**

리액트에서 Hooks 패턴을 사용하면 클래스형 컴포넌트를 작성하지 않고도 대부분의 기능을 구현할 수 있다. 이러한 기능이 리액트에 릴리즈 되었다고 해서 기존의 사용방식이 잘못된 것은 아니다. 리액트 매뉴얼에 따르면 기존의 클래스형 컴포넌트도 계속해서 지원될 예정이다. 그렇기 때문에 만약 유지 보수하고 있는 프로젝트에서 클래스형 컴포넌트를 사용하고 있다면 이를 굳이 함수형 컴포넌트와 Hooks를 사용하는 형태로 전환할 필요는 없다. 다만 매뉴얼에서는 새로 작성하는 컴포넌트의 경우 함수형 컴포넌트와 Hooks를 사용할 것을 권장하고 있다.

# 7장 컴포넌트 스타일링

**리엑트에서 DOM 요소에 스타일을 적용할 때는 문자열 형태로 넣는 것이 아니라 객체 형태로 넣어 주어야 한다. 스타일 이름 중에 - 문자가 포함되는 있으면 - 문자를 없애고 카멜 표기법으로 작성해야 한다. 따라서 background-color는 backgroundColor로 작성한다.**

- **컴포넌트 스타일 적용(1)**

**[src]-[Hello.js]**

```
const Hello = () => {
    return(
        <div>
            <h1 style={{color:'pink', background:'black', border:'5px solid yellow', padding:'20px', opacity:0.5}}>
                Hello, World!
            </h1>
        </div>
    );
}

export default Hello;
```

- **컴포넌트 스타일 적용(2)**

**[src]-[Hello.js]**

```
const Hello = () => {
    const style ={
        color: 'pink',
        background: 'black',
        border: '5px solid yellow',
        padding: '20px',
        opacity: 0.5
    }

    return(
        <div>
            <h1 style={ style }> Hello </h1>
        </div>
    );
}

export default Hello;
```

- **컴포넌트 스타일 적용(3)**

**[src]-[Hello.module.css]**

```
.box {
    width: 200px; height: 50px; background-color: blue;
}
```

**[src]-[App.js]**

```
import './App.css';
import Hello from './Hello';
import style_hello from './Hello.module.css';

function App() {
    return (
        <div className="App">
            <Hello/>
            <div className={ style_hello.box }>App</div>
        </div>
    );
}

export default App;
```

[src]-[App.module.css]

```
.box {
    width: 100px;
    height: 100px;
    background-color: red;
}
```

[src]-[App.css]

```
import './App.css';
import Hello from './Hello';
import style_hello from './Hello.module.css';
import style_app from './App.module.css';

function App() {
    return (
        <div className="App">
            <Hello/>
            <div className={style_hello.box}>Hello</div>
            <div className={style_app.box}>App</div>
        </div>
    );
}

export default App;
```

- **반응형 디자인**

**브라우저의 가로 크기에 따라 다른 스타일을 적용하기 위해서는 일반 CSS를 사용할 때와 똑같이 media 쿼리를 사용하면 된다.**

[src]-[App.css]

```
/* 기본적인 가로 크기 1024px에 가운데 정렬을 한다. */
.box {
    background: black;
    margin: 0px auto;
    width: 1024px;
    padding: 1rem;
}

/* 브라우저의 넓이가 1024px 이하이면 카로 크기를 768px 크기로 줄인다. */
@media screen and (max-width: 1024px) {
    .box { width: 768px; }
}

/* 브라우저의 넓이가 768이하이면 가로 크기를 꽉 채운다. */
@media screen and (max-width: 768px) {
    .box { width: 100%; }
}

.button {
    background: none;
    border: 2px solid white;
    border-radius: 4px;
    color: white;
    padding: 0.5rem;
    font-weight:600;
}
```

[src]-[StyledComponent.js]

```
const StyledComponent = () => {
    return(
        <div className="box">
            <button className="button">안녕하세요</button>
        </div>
    );
};

export default StyledComponent;
```

# 8장 일정관리 웹 애플리케이션 만들기

**필요한 라이브러리 설치 : 다양하고 예쁜 아이콘을 사용할 수 있는 라이브러리이다. 사용법은 http://react-icons.netlify.com에서 확인 가능하다.**

```
yarn add react-icons
```

- **UI 구성하기**

- TodoTemplage: 화면을 가운데 정렬시켜 주며 앱 타이틀을 보여준다. children으로 내부 JSX를 props로 받아 와서 렌더링해준다.
- TodoInsert: 새로운 항목을 입력하고 추가할 수 있는 컴포넌트이다. state를 통해 input의 상태를 관리한다.
- TodoList: todos 배열을 props로 받아 온 후 이를 배열 내장 함수 map을 사용해서  여러 개의 TodoListItem 컴포넌트로 변환하여 보여준다.
- TodoListItem: 각 할 일 항목에 대한 정보를 보여 주는 컴포넌트이다. todo 객체를 props로 받아 와서 상태에 따라 다른 스타일의 UI를 보여준다.

- **TodoTemplate 만들기**

[src]-[component]-[TodoTemplate.js]

```
import './TodoTemplate.css';

const TodoTemplate = ({ children }) => {
    return(
        <div className="TodoTemplate">
            <div className="app-title">일정 관리</div>
            <div className="content">{ children }</div>
        </div>
    );
};
export default TodoTemplate;
```

[src]-[component]-[TodoTemplate.css]

```
body{
    margin: 0px;
    padding: 0px;
    background: #e9ecef;
}
.TodoTemplate {
    width: 512px;
    /* width가 주어진 상태에서 좌우 중앙 정렬 */
    margin: 0px auto;
    margin-top: 6rem;
    border-radius: 4px;
    overflow: hidden;
}
.app-title {
    background: #22b8cf;
    color: white;
    height: 4rem;
    font-size: 1.5rem;
    /* 내용물의 세로 가운데 정렬 */
    align-items: center;
    /* 내용물의 가로 가운데 정렬 */
    justify-content: center;
    display: flex;
}
.content {
    background: white;
}
```


[src]-[App.js]

```
import TodoTemplate from "./component/TodoTemplate";

const App = () => {
    return(
        <TodoTemplate>Todo App을 만들자 </TodoTemplate>
    );
};
export default App;
```

- **TodoInsert 만들기**

**http://react-icons.netlify.com/#/icons/md 페이지로 들어가면 수많은 아이콘들과 이름이 함께 나타난다. 여기서 사용하고 싶은 아이콘을 고른 다음, import 구문을 사용하여 불러온 후 컴포넌트처럼 사용한다.**

[src]-[component]-[TodoInsert.js]

```
import { MdAdd } from 'react-icons/md';

const TodoInsert = () => {
    return(
        <form className="TodoInsert">
            <input placeholder="할 일을 입력 하세요"/>
            <button type="submit"><MdAdd/></button>
        </form>
    )
}

export default TodoInsert;
```

[src]-[App.js]

```
import TodoTemplate from "./component/TodoTemplate";
import TodoInsert from "./component/TodoInsert";

const App = () => {
    return(
        <TodoTemplate>
            <TodoInsert/>
        </TodoTemplate>
    );
};

export default App;
```

[src]-[component]-[TodoTemplate.css]

```
.TodoInsert {
    display: flex;
    background: #495057;
}

/* 기본 스타일 초기화 */
input {
    background: none;
    outline: none;
    border: none;
    padding: 0.5rem;
    font-size: 1.125rem;
    line-height: 1.5;
    color: white;
    /* 버튼을 제외한 영역을 모두 차지하기 */
    flex: 1;
}

input::placeholder{
    color: #dee2e6;
}

/* 기본 스타일 초기화 */
button {
    background: none;
    outline: none;
    border: none;
    background: #868e96;
    color: white;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1.5rem;
    align-items: center;
    cursor: pointer;
}
```

- **TodoListItem과 TodoList 만들기**

[src]-[component]-[TodoListItem.js]

```
import {
    MdCheckBoxOutlineBlank,
    MdRemoveCircleOutline,
    MdCheckBox } from "react-icons/md"

const TodoListItem = ({ todo }) => {
    const { text, checked } = todo;

    return(
        <div className="TodoListItem">
            <div className={ checked ? "checkbox_checked": "checkbox" }>
                { checked ? <MdCheckBox/> : <MdCheckBoxOutlineBlank/> }
                <div className="text">{ text }</div>
            </div>
            <div className="remove">
                <MdRemoveCircleOutline/>
            </div>
        </div>
    );
};

export default TodoListItem;
```

[src]-[component]-[TodoTemplate.css]

```
.TodoList {
    min-height: 320px;
    max-height: 513px;
    overflow: auto;
}

.TodoListItem {
    padding: 1rem;
    display: flex;
    align-items: center;
    /* 엘리먼트 사이사이에 테두리를 넣어 줌 */
    border-top: 1px solid #dee2e6;
}

.TodoListItem:nth-child(even){
    background: #f8f9fa;
}

svg {  /* 아이콘 */
    font-size: 1.5rem;
    margin-right: 0.5rem;
}

.checkbox, .checkbox_checked {
    cursor: pointer;
    /* 차지할 수 있는 영역 모두 차지 */
    flex: 1;
    display: flex;
    align-items: center;
}

.checkbox_checked svg{
    color: #22b8cf;
}

.checkbox_checked .text {
    color: #adb5bd;
    text-decoration: line-through;
}

.remove {
    cursor: pointer;
    color: #ff6b6b;
}
```

[src]-[component]-[TodoList.js]

```
import TodoListItem from "./TodoListItem"

const TodoList = ({ todos }) => {
    return (
        <div className="TodoList">
            { todos.map(todo => (
                <TodoListItem todo={todo} key={todo.id}/>
            )) }
        </div>
    );
};

export default TodoList;
```

[src]-[App.js]

```
import TodoTemplate from "./component/TodoTemplate";
import TodoInsert from "./component/TodoInsert";
import TodoList from "./component/TodoList";
import { useState } from "react";

const App = () => {
    const [todos, setTodos] = useState([
        { id: 1, text: '리액트의 기초 알아보기', checked: true },
        { id: 2, text: '컴포넌트 스타일링해 보기', checked: true },
        { id: 3, text: '일정 관리 앱 만들어 보기', checked: false }
    ]);

    return(
        <TodoTemplate>
            <TodoInsert/>
            <TodoList todos={ todos }/>
        </TodoTemplate>
    );
};

export default App;
```

- **항목 추가 기능 구현하기**

[src]-[component]-[TodoInsert.js]

```
import { useCallback, useState } from 'react';
import { MdAdd } from 'react-icons/md';

const TodoInsert = ({ onInsert }) => {
    const [value, setValue] = useState('');
    const onChange = useCallback(e=> {
        setValue(e.target.value);  //리렌더링될 때마다 함수를 새로 만드는 것이 아니라 한번 만들고 재사용한다.
    }, []);

    const onSubmit = useCallback(e => {
        onInsert(value);
        setValue('');  //value값 초기화
        e.preventDefault();  //submit 이벤트는 새로고침이 발생하므로 이를 방지한다.
    }, [onInsert, value]);

    return(
        <form className="TodoInsert" onSubmit={onSubmit}>
            <input placeholder="할 일을 입력하세요" value={value} onChange={onChange}/>
            <button type="submit"><MdAdd/></button>
        </form>
    )
}

export default TodoInsert;
```

[src]-[App.js]

```
import TodoTemplate from "./component/TodoTemplate";
import TodoInsert from "./component/TodoInsert";
import TodoList from "./component/TodoList";
import { useCallback, useState, useRef } from "react";

const App = () => {
    const [todos, setTodos] = useState([
        { id: 1, text: '리액트의 기초 알아보기', checked: true },
        { id: 2, text: '컴포넌트 스타일링해 보기', checked: true },
        { id: 3, text: '일정 관리 앱 만들어 보기', checked: false }
    ]);

    //id는 렌더링정보가 아니기 때문에 useState가아니라 useRef를 사용한다.
    const nextId = useRef(4);

    const onInsert = useCallback(text => {
        const todo = {
            id: nextId.current,
            text,
            checked: false
        };
        setTodos(todos.concat(todo));
        nextId.current += 1;  //nextId 1씩 더하기
    }, [todos]);

    return(
        <TodoTemplate>
            <TodoInsert onInsert={ onInsert }/>
            <TodoList todos={ todos }/>
        </TodoTemplate>
    );
};

export default App;
```

- **지우기 기능 구현하기**

[src]-[App.js]

```
...
const App = () => {
    ...
    const onRemove = useCallback( id => {
        setTodos(todos.filter(todo => todo.id !== id));
    }, [todos]);

    return(
        <TodoTemplate>
            <TodoInsert onInsert={ onInsert }/>
            <TodoList todos={ todos } onRemove={ onRemove }/>
        </TodoTemplate>
    );
};
```

[src]-[component]-[TodoList.js]

```
const TodoList = ({ todos, onRemove }) => {
    return (
        <div className="TodoList">
            { todos.map(todo => (
                <TodoListItem
                    todo={ todo }
                    key={ todo.id }
                    onRemove={ onRemove }/>
            )) }
        </div>
    );
};
```

[src]-[component]-[TodoListItem.js]

```
const TodoListItem = ({ todo, onRemove }) => {
    const { id, text, checked } = todo;

    return(
        <div className="TodoListItem">
            <div className={checked ? "checkbox_checked": "checkbox"}>
                {checked ? <MdCheckBox/> :<MdCheckBoxOutlineBlank/>}
                <div className="text"> { text }</div>
            </div>
            <div className="remove" onClick={()=>onRemove(id)}>
                <MdRemoveCircleOutline/>
            </div>
        </div>
    );
};
```

- **수정 기능**

[src]-[App.js]

```
const App = () => {
    ...
    const onToggle = useCallback(id=>{
        setTodos(todos.map(todo=> todo.id === id ? { ...todo, checked:!todo.checked } : todo));
    }, [todos]);

    return(
        <TodoTemplate>
            <TodoInsert onInsert={ onInsert }/>
            <TodoList todos={ todos } onRemove={ onRemove } onToggle={ onToggle }/>
        </TodoTemplate>
    );
};
```

[src]-[component]-[TodoList.js]

```
const TodoList = ({ todos, onRemove, onToggle }) => {
    return (
        <div className="TodoList">
            { todos.map(todo => (
                <TodoListItem
                    todo={ todo }
                    key={ todo.id }
                    onRemove={ onRemove }
                    onToggle={ onToggle }/>
            )) }
        </div>
    );
};
```

[src]-[component]-[TodoListItem.js]

```
const TodoListItem = ({ todo, onRemove, onToggle }) => {
    const {id, text, checked} = todo;

    return(
        <div className="TodoListItem">
            <div className={ checked ? "checkbox_checked": "checkbox" } onClick={()=>onToggle(id) }>
                { checked ? <MdCheckBox/> :<MdCheckBoxOutlineBlank/> }
                <div className="text"> { text }</div>
            </div>
            <div className="remove" onClick={ ()=>onRemove(id) }>
                <MdRemoveCircleOutline/>
            </div>
        </div>
    );
};
```

# 9장 리액트 라우터로 SPA 개발하기

SPA는 Single Page Application의 약어이다. 말 그대로 한 개의 페이지로 이루어진 애플리케이션이라는 의미이다. 리액트 같은 라이브러리 혹은 프레임워크를 사용하여 뷰 렌더링을 사용자의 브라우저가 담당하도록 하고 우선 애플리케이션을 브라우저에 불러와서 실행시킨 후에 사용자와의 인터랙션이 발생하면 필요한 부분만 자바스크립트를 사용하여 업데이트해 준다. 만약 새로운 데이터가 필요하면 서버 API를 호출하여 필요한 데이터만 새로 불러와 애플리케이션에서 사용한다.

- SPA의 단점

SPA의 단점은 앱의 규모가 커지면 자바스크립트의 파일이 너무 커진다는 것이다. 이것은 코드 스플리팅을 사용하면 라우트별로 파일들을 나누어서 트래픽과 로딩속도를 개선할 수 있다. 또한 자바스크립트를 실행하지 않는 일반 크롤러에서는 페이지의 정보를 제대로 수집하지 못하는 단점이 있지만 이것도 서버 사이드 렌더링을 통해 모두 해결할 수 있다.

- Router 라이브러리 설치

```
yarn add react-router-dom@5.3.0
```

- 프로젝트에 라우터 적용

프로젝트에 리액트 라우터를 적용할 때는 src/index.js 파일에서 react-router-dom에 내장되어 있는 BrowserRouter라는 컴포넌트를 사용하여 감싸준다. 이 컴포넌트는 웹 애플리케이션에 HTML5의 History API를 사용하여 페이지를 '새로고침'하지 않고도 주소를 변경하고 현재 주소에 관련된 정보를 props로 쉽게 조회하거나 사용할 수 있도록 해준다.

[src]-[index.js]

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>,
    document.getElementById('root')
);
reportWebVitals();
```

- 페이지 만들기

사용자가 웹 사이트에 들어왔을 때 맨 처음 보여줄 Home 컴포넌트와 웹 사이트를 소개하는 About 컴포넌트를 만든다.

[src]-[component]-[Home.js]

```
const Home = () => {
    return(
        <div>
            <h1>홈</h1>
            <p>가장 먼저 보여주는 홈페이지</p>
        </div>
    );
};
export default Home;
```

[src]-[component]-[About.js]

```
const About = () => {
    return(
        <div>
            <h1>소개</h1>
            <p>이 프로젝트는 리액트 라우터 기초를 실습해 보는 예제 프로젝트입니다.</p>
        </div>
    );
};
export default About;
```

- **Route 컴포넌트로 특정 주소에 컴포넌트 연결**

/about 경로로 들어가면 About 컴포넌트만 나오기를 기대했지만 예상과 다르게 두 컴포넌트가 모두 나타난다. /about 경로가 /규칙에도 일치하기 때문에 발생한 현상이다. 이를 수정하려면 Home을 위한 Route 컴포넌트를 사용할 때 exact라는 props를 true로 설정한다.

[src]-[App.js]

```
import { Route } from "react-router"
import About from './component/About';
import Home from './component/Home'

const App = () => {
    return(
        <div>
            <Route path="/" component={Home} exact={true}/>
            <Route path="/about" component={About}/>
        </div>
    );
};
export default App;
```

- **Link 컴포넌트를 사용하여 다른 주소로 이동하기**

Link 컴포넌트를 사용하여 페이지를 전환하면 페이지를 새로 불러오지 않고 애플리케이션은 그대로 유지한 상태에서 HTML5 History API를 사용하여 페이지의 주소만 변경해 준다. Link 컴포넌트 자체는 a 태그로 이루어져 있지만 페이지 전환을 방지하는 기능이 내장되어 있다.

[src]-[App.js]

```
import { Route } from "react-router"
import { Link } from "react-router-dom";
import About from './component/About';
import Home from './component/Home'

const App = () => {
    return(
        <div>
            <ul>
                <li><Link to="/">홈</Link></li>
                <li><Link to="/about">소개</Link></li>
            </ul>
            <hr/>
            <Route path="/" component={Home} exact={true}/>
            <Route path="/about" component={About}/>
        </div>
    );
};
export default App;
```

- **Route 하나에 여러 개의 path 설정하기**

Route를 두 번 사용하는 대신, path props를 배열로 설정해 주면 여러 경로(/about, /info,)에서 같은 컴포넌트를 보여 줄 수 있다.

[src]-[App.js]

```
import { Route } from "react-router"
import { Link } from "react-router-dom";
import About from './component/About';
import Home from './component/Home'
const App = () => {
    return(
        <div>
            <ul>
                <li><Link to="/home">홈</Link></li>
                <li><Link to="/about">소개</Link></li>
            </ul>
            <hr/>
            <Route path="/" component={Home} exact={true}/>
            <Route path={ ["/about", "/info"] } component={About}/>
        </div>
    );
};
export default App;
```

- **URL 파라미터와 쿼리**

**일반적으로 파라미터는 ID나 이름을 사용하여 조회할 때 사용하고 쿼리는 키워드를 검색하거나 페이지에 필요한 옵션을 전달할 때 사용한다.**

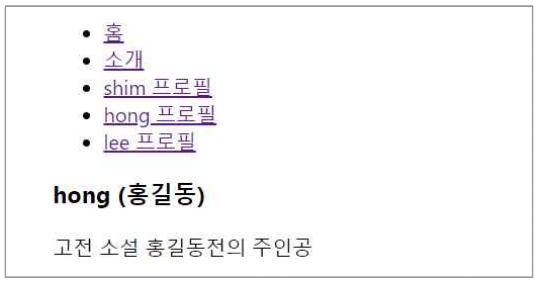
- **파라미터 예시: /profile/velopert**
- **쿼리 예시: /about?details=true**

- **URL 파라미터**

**아래는 /profile/velopert와 같은 형식으로 뒷부분에 유동적인 username값을 넣어줄 때 해당 값을 props로 받아 와서 조회하는 방법이다.**

**[src]-[component]-[Profile.js]**

```
const data = {
    shim: {
        name: '심청이',
        description: '리액트를 좋아하는 개발자'
    },
    hong: {
        name: '홍길동',
        description: '고전 소설 홍길동전의 주인공'
    },
    lee: {
        name: '이순신',
        description: '조선시대 장군'
    }
};

const Profile = ({ match }) => {
    const { username } = match.params;
    const profile = data[username];
    if(!profile){
        return <div>존재하지 않는 사용자입니다.</div>
    }
    return (
        <div>
            <h3>{ username } ({ profile.name })</h3>
            <p>{ profile.description }</p>
        </div>
    );
};

export default Profile;
```




**URL 파라미터를 사용할 때는 라우터로 사용되는 컴포넌트에서 받아 오는 match라는 객체 안의 params 값을 참조한다. match 객체 안에는 현재 컴포넌트가 어떤 경로 규칙에 의해 보이는지에 대한 정보가 들어있다. Profile 라우트 path 규칙에는 /profile/:username이라고 넣어 주면 된다. 이렇게 설정하면 match.params.username값을 통해 현재 username값을 조회할 수 있다.**

**[src]-[App.js]**

```
import Profile from './component/Profile';

const App = () => {
    return(
        <div>
            <ul>
                <li><Link to="/home">홈</Link></li>
                <li><Link to="/about">소개</Link></li>
                <li><Link to="/profile/shim">shim 프로필</Link></li>
                <li><Link to="/profile/hong">hong 프로필</Link></li>
                <li><Link to="/profile/lee">lee 프로필</Link></li>
            </ul>
            <hr/>
            <Route path="/" component={Home} exact={true}/>
            <Route path={ ['about', '/info'] } component={About}/>
            <Route path="/profile/:username" component={Profile}/>
        </div>
    );
};

export default App;
```

- URL 쿼리

쿼리는 location 객체에 들어있는 search값에서 조회할 수 있다. location 객체는 라우트로 사용된 컴포넌트에 props로 전달되며 웹 애플리케이션의 현재 주소에 대한 정보를 지니고 있다. 아래 location 객체는 http://localhost:3000/about?detail=true 주소로 들어갔을 때의 값이다. URL 쿼리를 읽을 때는 위 객체가 지난 값 중에서 search 값을 확인해야 한다. 이 값은 문자열 형태로 되어 있다. 문자열에 여러 가지 값을 설정할 경우에는 ?detail=true&another=1과 같이 &연산자로 연결해준다.

```
{
    "pathname": "/about",
    "search": "?detail=true",
    "hash": ""
}
```

search값에서 특정 값을 읽어 오기 위해서는 쿼리 문자열을 객체 형태로 변환해 주는 qs라는 라이브러리를 아래와 같이 설치해 주어야한다.

```
yarn add qs
```

쿼리를 사용할 때는 쿼리 문자열을 객체로 파싱하는 과정에서 결과 값은 언제나 문자열이다. 그렇기 때문에 숫자를 받아 와야 하면 parseInt 함수를 통해 꼭 숫자로 변환해 주고 지금처럼 논리 자료형 값을 사용해야 하는 경우에는 정확히 "true" 문자열이랑 일치하는지 비교한다. 결과를 확인하기위해 브라우저에서 http://localhost:3000/about?detail=true 주소로 직접 들어간 후 접속해 본다.

[src]-[component]-[About.js]

```
import qs from "qs";

const About = ({ location }) => {
    const query = qs.parse(
        location.search, {
            ignoreQueryPrefix: true  //이 설정을 통해 문자열 맨 앞의 ?를 생략한다.
        }
    );
    const showDetaile = query.detail === 'true';  //쿼리의 파싱 결과 값은 문자열이다.

    return(
        <div>
            <h1>소개</h1>
            <p>이 프로젝트는 리액트 라우터 기초를 실습해 보는 예제 프로젝트입니다.</p>
            { showDetaile && <p>detaile 값을 true로 설정하셨군요!</p> }
        </div>
    );
};
export default About;
```

- 서브 라우트

서브 라우트는 라우트 내부에 또 라우트를 정의하는 것을 의미한다. 아래는 프로필 링크를 보여 주는 Profiles라는 라우트 컴포넌트를 따로 만들고 그 안에서 Profile 컴포넌트를 서브 라우트로 사용하도록 작성한다.

[src]-[component]-[Profiles.js]

```
import { Link, Route } from "react-router-dom"
import Profile from "./Profile";

const Profiles = () => {
    return(
        <div>
            <h3>사용자 목록</h3>
            <ul>
                <li><Link to="/profiles/shim">심청이</Link></li>
                <li><Link to="/profiles/hong">홍길동</Link></li>
                <li><Link to="/profiles/lee">이순신</Link></li>
            </ul>
            <Route path="/profiles" exact render={()=><div>사용자를 선택해 주세요!</div>}/>  //exact와 exact={true}는 같은 의미이다.
            <Route path="/profiles/:username" component={Profile}/>
        </div>
    );
};
export default Profiles;
```

첫 번째 Route 컴포넌트 component 대신 render라는 props를 넣어 주었다. 컴포넌트 자체를 전달하는 것이 아니라 보여 주고 싶은 JSX를 넣어 줄 수 있다. 컴포넌트를 넣어주기 애매한 상황이나 컴포넌트에 props를 별도로 넣어 주고 싶을 때도 사용할 수 있다.

App 컴포넌트에 있던 프로필 링크를 지우고 Profiles 컴포넌트를 /profiles 경로에 연결시켜 준다.

[src]-[App.js]

```
...
import Profiles from "./Profiles";

function App() {
    return(
        <div>
            <ul>
                <li><Link to="/">홈</Link></li>
                <li><Link to="/about">소개</Link></li>
                <li><Link to="/profiles">프로필</Link></li>
            </ul>
            <hr/>
            <Route path="/" component={Home} exact={true}/>
            <Route path={['/about','/info']} component={About}/>
            <Route path="/profiles" component={Profiles}/>
        </div>
    );
};
export default App;
```


## • 리액트 라우터 부가 기능

• history

history 객체는 라우트로 사용된 컴포넌트에 math, location과 함께 전달되는 props중 하나로 이 객체를 통해 컴포넌트 내에 구현하는 메서드에서 라우터 API를 호출할 수 있다. 예를 들어 특정 버튼을 눌렀을 때 뒤로 가거나 로그인 후 화면을 전환하거나 다른 페이지로 이탈하는 것을 방지해야 할 때 history를 활용한다. 아래는 history를 사용한 예제이다.

[src]-[component]-[HistorySample.js]

```
import { Component } from "react";

class HistorySample extends Component{
    handleGoBack = () => {  //뒤로가기
        this.props.history.goBack();
    }

    handleGoHome = () => {  //홈으로 이동
        this.props.history.push('/');
    }

    componentDidMount() {
        //페이지 변화가 생기려고 할 때마다 정말 나갈 것인지를 질문함
        this.unblock = this.props.history.block('정말 떠나실 건가요?');
    }

    componentWillUnmount() {
        //컴포넌트가 언마운트되면 질문을 멈춤
        if(this.unblock){
            this.unblock();
        }
    }

    render(){
        return(
            <div>
                <button onClick={this.handleGoBack}>뒤로</button>
                <button onClick={this.handleGoHome}>홈으로</button>
            </div>
        );
    }
}
export default HistorySample;
```

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

const HistorySample = ({history}) => {
    const onGoHome = () => {
        history.push('/');
    }

    const onGoBack = () => {
        history.goBack();
    }

    useEffect(() => {
        console.log(history);
        const unblock = history.block('정말 떠나 실건가요?');
        return () => {
          unblock();
        };
      }, [history]);

    return (
        <div>
            <button onClick={onGoBack}>뒤로</button> 
            <button onClick={onGoHome}>홈으로</button>
        </div>
    );
};

export default HistorySample;
```

App에서 /history 경로에 해당 컴포넌트가 보이도록 설정한다. http://localhost:300/history에서 버튼의 동작을 확인한다.

[src]-[App.js]

```
import HistorySample from './HistorySample';

function App() {
    return(
        <div>
            <ul>
                <ul>
                <li><Link to="/">홈</Link></li>
                <li><Link to="/about">소개</Link></li>
                <li><Link to="/profiles">프로필</Link></li>
                <li><Link to="/history">History 예제</Link></li>
            </ul>
            <hr/>
            ...
            <Route path="/history" component={HistorySample}/>
        </div>
    );
};
```

- withRouter

withRouter 함수는 HoC(Higher-order Component)이다. 라우터로 사용된 컴포넌트가 아니어도 match, location, history 객체를 접근할 수 있게 해준다. withRouter를 사용할 때는 컴포넌트를  내보내 줄 때 함수로 감싸준다. JSON.stringify의 두 번째 파라미터와 세 번째 파라미터를 위와 같이 null, 2로 설정해 주면 들여쓰기가 적용된 상태로 문자열이 만들어진다.

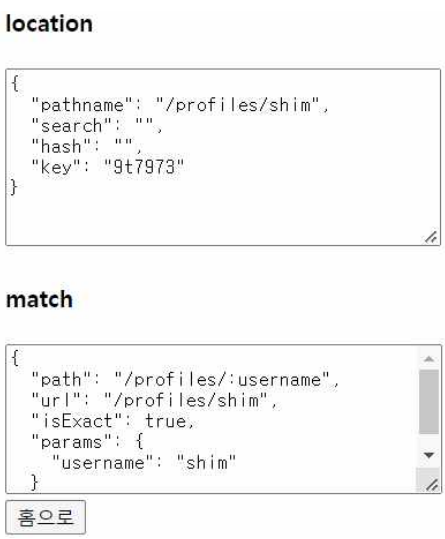
[src]-[component]-[WithRouterSample.js]

```
import { withRouter } from "react-router";
const WithRouterSample = ({ location, match, history }) => {
    return(
        <div>
            <h4>location</h4>
            <textarea rows={7} readOnly={true} value={JSON.stringify(location, null, 2)}/>
            <h4>match</h4>
            <textarea rows={7} readOnly={true} value={ JSON.stringify(match, null, 2)}/>
            <div><button onClick={() => history.push('/') }>홈으로</button></div>
        </div>
    );
};
export default withRouter(WithRouterSample);
```

withRouter를 사용하면 현재 자신을 보여주고 있는 라우트 컴포넌트(현재 Profiles)를 기준으로 match가 전달된다. WithRouterSample 컴포넌트를 Profile 컴포넌트에 넣으면 match 쪽에 URL 파라미터가 제대로 보일 것이다.

[src]-[component]-[Profile.js]

```
mport { withRouter } from "react-router";
import WithRouterSample from "./WithRouterSample";
...
const Profile = ({ match }) => {
    const { username } = match.params;
    const  profile = data[username];

    if(!profile){
        return <div>존재하지 않는 사용자입니다.</div>
    }
    return(
        <div>
            <h3>{username}({profile.name})</h3>
            <p>{profile.description}</p>
            <WithRouterSample/>
        </div>
    );
};

export default withRouter(Profile);
```



- Switch

switch 컴포넌트는 여러 Route를 감싸서 그 중 일치하는 단 하나의 라우트만을 렌더링 시켜 준다. Switch를 사용하면 모든 규칙과 일치하지 않을 때 보여 줄 Not Found 페이지도 구현할 수 있다. 존재하지 않는 페이지인 http://localhost:3000/nowhere로 접속해 보자.

[src]-[App.js]

```
import { Route, Switch } from 'react-router';
...
function App() {
    return(
        <div>
            <ul>
                <li><Link to="/">홈</Link></li>
                <li><Link to="/about">소개</Link></li>
                <li><Link to="/profiles">프로필</Link></li>
                <li><Link to="/history">History 예제</Link></li>
            </ul>
            <hr/>
            <Switch>
                <Route path="/" component={Home} exact={true}/>
                <Route path={['/about','/info']} component={About}/>
                <Route path="/profiles" component={Profiles}/>
                <Route path="/history" component={HistorySample}/>
                <Route render={ ({location}) => (  //path를 따로 정의하지 않으면 모든 상황에 렌더링됨
                    <div>
                        <h2>이 페이지는 존재하지 않습니다.</h2>
                        <p>{location.pathname}</p>
                    </div>
                )}/>
            </Switch>
        </div>
    );
};

export default App;
```

- NavLink

NavLink는 Link와 비슷하다. 현재 경로와 Link에서 사용하는 경로가 일치하는 경우 특정 스타일 혹은 CSS 클래스를 적용할 수 있는 컴포넌트이다. NavLink에서 링크가 활성화되었을 때의 스타일을 적용할 때는 activeStyle 값을 CSS 클래스를 적용할 때는 activeClassName 값을 props로 넣어 주면 된다. profiles에서 사용하고 있는 컴포넌트에서 Link 대신 NavLink를 사용하게 하고, 현재 선택되어 있는 경우 검정색 배경에 흰색 글씨로 스타일을 보여 주게끔 코드를 수정한다. 사용자 목록에 있는 링크를 클릭했을 때 색상이 바뀐다.

[src]-[component]-[Profiles.js]

```
mport { NavLink, Route } from "react-router-dom"
import Profile from "./Profile";

const Profiles = () => {
    const activeStyle = {
        background: 'black',
        color: 'white'
    };

    return(
        <div>
            <h3>사용자 목록</h3>
            <ul>
                <li><NavLink activeStyle={activeStyle} to="/profiles/shim">심청이</NavLink></li>
                <li><NavLink activeStyle={activeStyle} to="/profiles/hong">홍길동</NavLink></li>
                <li><NavLink activeStyle={activeStyle} to="/profiles/lee">이순신</NavLink></li>
            </ul>
            <Route path="/profiles" exact render={()=><div>사용자를 선택해 주세요</div>}/>
            <Route path="/profiles/:username" component={Profile}/>
        </div>
    );
};

export default Profiles;
```

# 10장 외부 API를 연동하여 뉴스 뷰어 만들기

- **비동기 작업의 이해**

서버의 API를 사용해야 할 때는 네트워크 송수신 과정에서 시간이 많이 걸리기 때문에 작업이 즉시 처리되는 것이 아니라 응답을 받을 때까지 기다렸다가 전달받은 응답 데이터를 처리한다. 이 과정에서 해당 작업을 비동기적으로 처리하게 된다. 만약 작업을 동기적으로 처리한다면 요청이 끝날 때까지 기다리는 동안 중지 상태가 되기 때문에 다른 작업을 할 수 없다. 하지만 이름 비동기적으로 처리한다면 웹 애플리케이션은 멈추지 않기 때문에 동시에 여러 가지 요청을 처리할 수도 있고 기다리는 과정에서 다른 함수도 호출할 수 있다.

- **콜백 함수**

자바스크립트에서 비동기 작업을 할 때 가장 흔히 사용하는 방법이 콜백 함수를 사용하는 것이다. 아래 코드에서는 printMe가 3초 뒤에 호출되도록 printMe 함수 자체를 setTimeout 함수의 인자로 전달해 주었는데, 이런 함수를 콜백 함수라고 부른다.

printMe 콜백 함수 예제

```
function printMe() {
    console.log('Hello World');
}

setTimeout(printMe, 3000);
console.log('대기중...');
```

- **async/await**

async와 await는 자바스크립트의 비동기 처리 패턴 중 가장 최근에 나온 문법입니다. 기존의 비동기 처리 방식인 콜백 함수와 프로미스의 단점을 보완하고 개발자가 읽기 좋은 코드를 작성할 수 있게 도와준다. 이 문법을 사용하려면 함수의 앞부분에 async 키워드를 추가하고 해당 함수 내부에서 Promise의 앞부분에 await 키워드를 사용한다.

- **axios 예제 실습**

axios는 현재 가장 많이 사용되고 있는 자바스크립트 HTTP 클라이언트이다. 이 라이브러리의 특징은 HTTP 요청을 promise 기반으로 처리한다는 점이다. 이 라이브러리를 사용하기 위해 아래와 같이 설치한다.

```
yan add axios;
```

[src]-[App.js]

```
import axios from "axios";
import { useState } from "react";

const App = () => {
    const [data, setData] = useState(null);
    const onClick = () => {
        axios.get('https://jsonplaceholder.typicode.com/todos/1')
        .then(response => {  //결과를 .then을 통해 비동기적으로 확인한다.
            setData(response.data)
        });
    };
    return(
        <div>
            <div><button onClick={onClick}>불려오기</button></div>
            { data && <textarea rows={7} value={ JSON.stringify(data, null, 2) } readOnly/> }
        </div>
    );
};
export default App;
```

```
callAPI = () => {
    fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then(res => res.json())
    .then(json => {
        this.setState({
            data: json.title
        });
    });
}
```

[src]-[App.js]

```
const onClick = async() => {
    try {
        const response = await axios.get('https://jsonplaceholder.typicode.com/todos/1');
        setData(response.data);
    }catch(e){
        console.log(e);
    }
};
```

- **newsapi API키 발급 받기**

**이번 프로젝트에서는 newsapi에서 제공하는 API를 사용하여 최신 뉴스를 불러온 후 보여 줄 것이다. 이를 수행하기 위해서는 사전에 newsapi에서 API 키를 발급 받아야 한다. API 키는 https://newsapi.org/register에 가입하여 발급 받는다. 사용할 API 주소는 두 가지 형태이다.**

**- 전체 뉴스 불러오기**

```
GET https://newsapi.org/v2/top-headlines?country=kr&apiKey=0d2cfc3452514a4f93903b35e762e40d
```

**- 특정 카테고리 뉴스 불러오기: 여기서 카테고리는 business, entertainment, health, science, sports, technology 중에 골라서 사용할 수 있다.**

```
GET https://newsapi.org/v2/top-headlines?country=kr&category=business&apiKey=0d2cfc3452514a4f93903b35e762e40d
```

- **전체 뉴스 불러오기**

**[src]-[App.js]**

```
import axios from "axios";
import { useState } from "react";

const App = () => {
    const [data, setData] = useState(null);
    const onClick = async() => {
        try {
            const response = await axios.get(
                'https://newsapi.org/v2/top-headlines?country=kr&apiKey=0d2cfc3452514a4f93903b35e762e40d'
            );
            setData(response.data);
        }catch(e) {
            console.log(e);
        }
    };

    return(
        <div>
            <div><button onClick={onClick}>불려오기</button></div>
            {data && <textarea rows={7} value={JSON.stringify(data, null, 2)} readOnly/>}
        </div>
    );
};

export default App;
```

```
const url = 'https://dapi.kakao.com/v3/search/book?target=title&query=자바';
const config = {
    headers:{"Authorization":"KakaoAK 696c16eaa4a3dfc09b778dd60435bb87"}
}
const res=await axios.get(url,config);
setBooks(res.data.documents);
```

- **뉴스 뷰어 UI 만들기**

**뉴스 데이터에는 제목(title), 내용(description), 링크(url), urlToImage(뉴스 이미지) 정보들로 이루어진 JSON 객체이다.**

- **NewsItem 만들기**

[src]-[components]-[NewsItem.js]

```
import "./NewsItem.css";

const NewsItem = ({article}) => {
    const { title, description, url, urlToImage } = article;

    return(
        <div className="newsItemBlock">
            <div className="thumbnail">
                <a href={url}>
                    <img src={urlToImage} alt="thumbnail"/>
                </a>
            </div>
            <div className="contents">
                <a href={url}><h2>{title}</h2></a>
                <p>{description}</p>
            </div>
        </div>
    );
};

export default NewsItem;
```

[src]-[component]-[NewsItem.css]

```
.newsItemBlock{
    display: flex;
    margin-top: 3rem;
}
.thumbnail img{
    width: 160px;
    height: 100px;
}
.contents {
    margin-left: 1rem;
}
.contents h2 {
    margin-top: 0px;
}
.contents a {
    color: black;
}
```

- **NewsList 만들기**

[src]-[components]-[NewsList.js]

```
import "./NewsList.css";
import NewsItem from "./NewsItem";

const sampleArticle ={
    title: '제목',
    description: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. ...',
    url: 'https://google.com',
    urlToImage: 'http://placehold.it/160x160'
};

const NewsList = () => {
    return(
        <div className="newsListBlock">
            <NewsItem article={sampleArticle}/>
            <NewsItem article={sampleArticle}/>
            <NewsItem article={sampleArticle}/>
        </div>
    );
};

export default NewsList;
```

[src]-[components]-[NewsList.css]

```
.newsListBlock {
    padding-bottom: 3rem;
    width: 768px;
    margin: 0px auto;
    margin-top: 2rem;

    @media screen and (max-width: 768px){
        width: 100%;
        padding-left: 1rem;
        padding-right: 1rem;
    }
}
```

- 데이터 연동하기

useEffect를 사용하여 컴포넌트가 처음 렌더링되는 시점에 API를 요청한다. 주의할 점은 useEffect에 등록하는 함수에 async를 붙이면 안 된다. useEffect에서 반환해야 하는 값은 뒷정리 함수이기 때문이다. 따라서 useEffect 내부에서 async/await를 사용하고 싶다면 함수 내부에 붙은 또 다른 함수를 만들어서 사용해 주어야 한다.

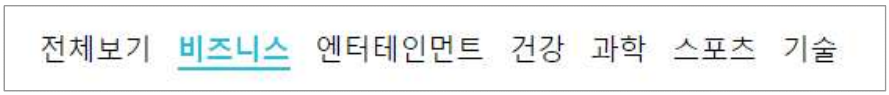
뉴스 데이터 배열을 map함수를 사용하여 컴포넌트 배열로 변환할 때 주의할 점이 있다. map 함수를 사용하기 전에 꼭 !articles를 조회하여 해당 값이 현재 null이 아닌지 검사해야 한다. 이 작업을 하지 않으면 아직 데이터가 없을 때 null에는 map 함수가 없기 때문에 렌더링 과정에서 오류가 발생한다. 그래서 애플리케이션이 제대로 나타나지 않고 흰 페이지만 보이게 된다.

[src]-[components]-[NewsList.js]

```
import "./NewsList.css";
import NewsItem from "./NewsItem";
import { useEffect, useState } from "react";
import axios from "axios";

const NewsList = () => {
    const [articles, setArticles] = useState(null);
    const [loading, setLoading] = useState(false);

    useEffect(()=> {
        //async를 사용하여 함수 따로 선언
        const fetchData = async() => {
            setLoading(true);
            try {
                const response = await axios.get( 'https://newsapi.org/v2/top-headlines?country=kr&apiKey=...');
                setArticles(response.data.articles);
            }catch(e) {
                console.log(e);
            }
            setLoading(false);
        };
        fetchData();
    }, []);  //컴포넌트가 화면에 맨 처음 렌더링될 때만 실행하고 업데이트될 때는 실행하지 않으려면 두 번째 파라미터로 비어 있는 배열을 넣어 준다.

    //대기 중일 때
    if(loading) {
        return <div className="newsListBlock">불러오는중...</div>
    }
    //아직 articles 값이 설정되지 않았을 때
    if(!articles) {
        return null;
    }
    //articles 값이 유효할 때
    return(
        <div className="newsListBlock">
            { articles.map(article => (
                <NewsItem key={article.url} article={article}/>
            )) }
        </div>
    );
};

export default NewsList;
```

- **카테고리 선택 UI 만들기**

**뉴스 카테고리는 비즈니스(business), 연예(Entertainment), 건강(health), 과학(Science), 스포츠(Sports), 기술(technology) 총 여섯 개이며 화면에 보여 줄 때는 영어로 된 값을 그대로 보여 주지 않고 아래 그림처럼 한글로 보여 준 뒤 클릭했을 때는 영어 값을 사용한다.**




[src]-[component]-[Categories.js]

```
import "./Categories.css";

const categories =[
    { name: 'all', text: '전체보기' },
    { name: 'business', text: '비즈니스' },
    { name: 'entertainment', text: '엔터테인먼트' },
    { name: 'health', text: '건강' },
    { name: 'science', text: '과학' },
    { name: 'sports', text: '스포츠' },
    { name: 'technology', text: '기술' },
];
const Categories = ({category, onSelect}) => {
    return(
        <div className="categoriesBlock">
            {categories.map(c => (
                <div className="category" key={c.name} onClick={()=>onSelect(c.name)}>
                    <span className={category===c.name ? 'active': ''}>{c.text}</span>
                </div>
            ))}
        </div>
    );
};

export default Categories;
```

[src]-[component]-[Categories.css]

```
.categoriesBlock{
    display: flex;
    width: 768px;
    margin: 0px auto;
    padding: 1rem;

    @media screen and (max-width: 768px){
        width: 100%;
        overflow: auto;
    }
}

.category {
    font-size: 1.125rem;
    cursor: pointer;
    text-decoration: none;
    color: inherit;
    padding-bottom: 0.25rem;
    margin-left: 1rem;
}

.category:hover {
    color: #b0b5bb;
}

.category .active {
    color: #22b8cf;
    border-bottom: 2px solid #22b8cf;
    font-weight: 600;
}

.category .active:hover {
    color: #abdfe6;
}
```

[src]-[App.js]

```
import { useCallback, useState } from "react";
import Categories from "./components/Categories";
import NewsList from "./components/NewsList";

const App = () => {
  const [category, setCategory] = useState('all');
  const onSelect = useCallback(category => setCategory(category), []); //렌더링할 때 만들었던 함수를 계속해서 재사용한다.

    return(
        <div>
            <Categories category={category} onSelect={onSelect}/>
            <NewsList category={category}/>
        </div>
    );
};

export default App;
```

- API를 호출할 때 카테고리 지정하기

NewsList 컴포넌트에서 현재 props로 받아 온 category에 따라 카테고리를 지정하여 API를 요청 하도록 구현한다. 추가로 category 값이 바뀔 때마다 뉴스를 새로 불러와야 하기 때문에 useEffect의 의존 배열(두번째 파라미터로 설정하는 배열)에 category를 넣어 주어야 한다. 만약 이 컴포넌트를 클래스형 컴포넌트로 만들게 된다면 componetDidMount와 componentDidUpdate에서 요청을 시작하도록 설정한다. 함수형 컴포넌트라면 useEffect 한 번으로 컴포넌트가 맨 처음 렌더링될 때 그리고 category 값이 바뀔 때 요청하도록 설정해 준다.

[src]-[components]-[NewsList.js]

```
import "./NewsList.css";
import NewsItem from "./NewsItem";
import { useEffect, useState } from "react";
import axios from "axios";

const NewsList = ({category}) => {
    const [articles, setArticles] = useState(null);
    const [loading, setLoading] = useState(false);

    useEffect(()=> {
        const fetchData = async() => {
            setLoading(true);
            try {
                const url = 'https://newsapi.org/v2/top-headlines?country=kr';
                const query = category === 'all' ? '' : '&category='+ category;
                const apiKey='&apiKey=0d2cfc3452514a4f93903b35e762e40d';
                const response = await axios.get(url + query + apiKey);
                setArticles(response.data.articles);
            }catch(e) {
                console.log(e);
            }
            setLoading(false);
        }
        fetchData();
    }, [category]);

    if(loading) {  //대기 중일 때
        return <div className="newsListBlock">불러오는중...</div>
    }
    if(!articles) {  //아직 articles 값이 설정되지 않았을 때
        return null;
    }
    return(  //articles 값이 유효할 때
        <div className="newsListBlock">
           { articles.map(article => (
                <NewsItem key={article.url} article={article}/>
           ))}
        </div>
    );
};

export default NewsList;
```

- **리액트 라우터 적용하기**

위에서 진행한 뉴스 뷰어 프로젝트에 리액트 라우트를 적용해 보겠다. 기존에는 카테고리 값을 useState로 관리하였는데 이번에는 이 값을 리액트 라우터의 URL 파라미터를 사용하여 관리해 보겠다. 우선 현재 프로젝트에 리액트 라우터를 설치하고 적용한다.

### 1) 리액트 라우터 설치

```
yarn add react-router-dom
```

### 2) 리액트 라우터 적용

[src]-[index.js]

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>,
    document.getElementById('root')
);

reportWebVitals();
```

### 3) NewsPage 생성

현재 선택한 category 값을 URL 파라미터를 통해 사용할 것이므로 Categories 컴포넌트에서 현재 선택한 카테고리 값을 알려 줄 필요도 없고, onSelect 함수를 따로 전달해 줄 필요도 없다. 새로운 NewsPage.js 파일을 생성하고 아래와 같이 코드를 작성한다.

[src]-[components]-[NewsPage.js]

```
import Categories from "./Categories"
import NewsList from "./NewsList"

const NewsPage = ({ match }) => {  //match객체를 보낸 path, url, params들이 저장되어있다.
    //카테고리가 선택되지 않았으면 기본값 all로 사용
    const category = match.params.category || 'all';

    return(
        <div>
            <Categories/>
            <NewsList category={category}/>
        </div>
    );
};

export default NewsPage;
```

### 4) App의 기존 내용을 지우고 Route를 정의해 준다.

아래 코드에서 사용된 path에 /:category?와 같은 형태로 맨 뒤에 물음표 문자가 들어가 있다. 이는 category 값이 선택적이라는 의미이다. 즉 있을 수도 있고 없을 수고 있다는 뜻이다. category URL 파라미터가 없다면 전체 카테고리를 선택한 것으로 간주한다.

[src]-[App.js]

```
import { Route } from "react-router";
import NewsPage from "./components/NewsPage";

const App = () => {
    return (
        <Route path="/:category?" component={NewsPage}/>;
    )
};

export default App;
```

5) Categories에서 NavLink 사용하기

NavLink로 만들어진 Category 컴포넌트에 to 값은 '/카테고리이름'으로 설정해 주었다. 그리고 카테고리 중에서 전체보기의 경우는 예외적으로 'all' 대신에 '/'로 설정했다. to 값이 '/'를 가리키고 있을 때는 exact 값을 true로 해주어야 한다.

[src]-[components]-[categories.js]

```
import { NavLink } from "react-router-dom";
import "./Categories.css";

const categories =[ ... ];

const Categories = ({category}) => {
    return(
        <div className="categoriesBlock">
            {categories.map(c => (
                <div className="category">
                    <NavLink
                        activeClassName="active"
                        exact={c.name === 'all'}
                        to={c.name === 'all' ? '/' : '/'+c.name}>
                        {c.text}
                    </NavLink>
                </div>
            ))}
        </div>
    );
};
export default Categories;
```

[src]-[components]-[Categories.css]

```
.category a{  /* 기존에 .category를 .category a로 수정 */
    font-size: 1.125rem;
    cursor: pointer;
    text-decoration: none;
    color: inherit;
    padding-bottom: 0.25rem;
    margin-left: 1rem;
}
```

- usePromise 커스텀 Hook 만들기

컴포넌트에서 API 호출처럼 Promise를 사용해야 하는 경우 간결하게 코드를 작성할 수 있도록 해주는 커스텀 Hook을 만들어서 적용하자.

[src]-[lib]-[usePromise.js]

```
import { useEffect, useState } from "react";

export default function usePrimise(promiseCreator, deps){  //파라미터 deps배열은 usePromise 내부의 useEffect의 의존 배열로 설정한다.
    //대기중, 완료, 실패에 대한 상태 관리
    const [loading, setLoading] = useState(false);
    const [resolved, setResolved] = useState(null);
    const [error, setError] = useState(null);

    useEffect(() => {
        const process = async () => {
            setLoading(true);
            try{
                const resolved = await promiseCreator();
                setResolved(resolved);
            }catch(e){
                setError(e);
            }
            setLoading(false);
        };
        process();
        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, deps);

    return[loading, resolved, error];
}
```

[src]-[components]-[NewsList.js]

```
import "./NewsList.css";
import NewsItem from "./NewsItem";
import axios from "axios";
import usePrimise from "../lib/usePromise";

const NewsList = ({category}) => {
    const [loading, response, error] =usePrimise(() => {
            const url = 'https://newsapi.org/v2/top-headlines?country=kr';
            const query = category === 'all' ? '' : '&category='+ category;
            const apiKey='&apiKey=0d2cfc3452514a4f93903b35e762e40d';
            return axios.get(url + query + apiKey);
        }, [category]
    );

    //아직 response 값이 설정되지 않았을 때
    if(loading) {
        return <div className="newsListBlock">불러오는중...</div>
    }

    //아직 articles 값이 설정되지 않았을 때
    if(!response) {
        return null;
    }

    //에러가 발생했을 때
    if(error){
        return <div className="newsListBlock">에러 발생!</div>
    }

    //response 값이 유효할 때
    const { articles } = response.data;
    return(
        <div className="newsListBlock">
            {articles.map(article => (
                <NewsItem key={article.url} article={article}/>
            ))}
        </div>
    );
};
export default NewsList;
```

- **최종결과**

# 11장 Context API

Context API는 리액트 프로젝트에서 전역적으로 사용할 데이터가 있을 때 유용한 기능이다. 이를테면 사용자 로그인 정보, 애플리케이션 환경 설정, 테마 등 여러 종류가 있다. 이 기능은 리액트 관련 라이브러리에서도 많이 사용되고 있다.

- React Context 없이 개발하기

먼저 ReactContext 없이 props만을 이용해서 전역 데이터를 여러 컴포넌트에 걸쳐서 접근할 수 있는지에 대해서 살펴보겠다. 아래 예제는 사용자가 설정한 언어에 따라서 영어 또는 한국어로 텍스트를 표시해 주는 앱이다.

[src]-[App.js]

```
import {useState} from 'react';
import Button from './Button';
import Title from './Title';
import Message from './Message';

function App() {
    const [lang, setLang] = useState('en')
    const onToggleLang = () => {
        const changLan = lang === 'en' ? 'kr' : 'en';
        setLang(changLan);
    };

    return(
        <div>
            <Button lang={lang} onToggleLang={onToggleLang}/>
            <Title lang={lang}/>
            <Message lang={lang}/>
        </div>
    );
}

export default App;
```

아래 3개의 하위 컴포넌트는 props로 넘어온 lang 값에 따라서 영어와 한국어의 텍스틀 렌더링한다. Button 컴포넌트는 onToggleLang 함수값을 onClick 핸들러에 설정해줌으로써 버튼 클릭 시 사용자 언어 설정값이 바뀌도록 한다.

[src]-[components]-[Button.js]

```
const Button = ({ lang, onToggleLang }) => {
    return <button onClick={onToggleLang}>{lang}</button>
};
export default Button;
```

[src]-[components]-[Title.js]

```
const Title = ({ lang }) => {
    const text = lang === 'en' ? 'Context' : '컨텍스트';
    return <h1>{text}</h1>
};
export default Title;
```

[src]-[componetns]-[Message.js]

```
import { Component } from "react";

class Message extends Component{
    render(){
        const {lang} = this.props;

        if(lang === 'en'){
            return <div><p>{lang}:Context provides a way to ...</p></div>
        }else {
            return <div><p>{lang}:컨텍스트는 ... 제공합니다.</p></div>
        }
    }
}

export default Message;
```

- **React Context를 사용해서 개발하기**

위에서 살펴본 것처럼 큰 규모의 앱에서는 전역 데이터를 관리하는데 좀 더 나은 방식이 필요하다. React Context는 전역 데이터를 좀 더 단순하지만 체계적인 방식으로 접근 할 수 있도록 도와준다.

### 1) Context생성하기

전역 데이터를 관리하기 위해서 React 패키지에서 제공하는 createContext라는 함수를 사용한다. React Context는 전역 데이터를 담고 있는 하나의 저장 공간이라고 생각할 수 있다. 아래는 createContext 함수의 인자로 해당 컨텍스트에 디폴트로 저장할 값을 넘긴다.

[src]-[contexts]-[LangContext.js]

```
import { createContext } from "react";
const LangContext = createContext('en');
export default LangContext;
```

### 2) Provider 사용

아래와 같이 컴포넌트에 Provider로 감싸주면 그 하위에 있는 모든 컴포넌트들은 이 ReactContext에 저장되어 있는 전역 데이터에 접근할 수 있다. value 속성값을 지정하지 않을 경우 Context를 생성할 때 넘겼던 디폴트값이 사용된다. 예전 앱에서는 모든 하위 컴포넌트에 props로 사용자 언어 설정값인 lang을 넘겨주었는데 바뀐 코드에서는 대신에 Provider로 한번 감싸고 value로 이 값을 설정해준다.

[src]-[App.js]

```
import { Component } from "react";
import Message from "./components/Message";
import Button   from './components/Button';
import Title from './components/Title';
import LangContext from "./contexts/LangContext";

class App extends Component {
    ...

    render() {
        const { lang } = this.state;

        return (
            <LangContext.Provider value={lang}>
                <Button lang={lang} onToggleLang = {this.onToggleLang }/>
                <Title lang={lang}/>
                <Message lang={lang}/>
            </LangContext.Provider>
        );
    }
}

export default App;
```

### 3) Consumer로 Context 접근하기

이번에는 lang의 값을 props로 받아오는 것이 아니라 LangContext 안에 들어 있는 Consumer라는 컴포넌트를 통해 색상을 조회한다. Consumer 사이에 중괄호를 열어서 그 안에 함수를 넣어 주었다. 이러한 패턴을 Function as child 혹은 Render Props라고 한다. 컴포넌트의 children이 있어야 할 자리에 일반 JSX 혹은 문자열이 아닌 함수를 전달한다.

[src]-[components]-[Title.js]

```
import LangContext from "../contexts/LangContext";

const Title = () => {
    return(
        <LangContext.Consumer>
            {(lang) => {
                const text = lang === 'en' ? 'Context' : '컨텍스트';
                return <h1>{text}</h1>
            }}
        </LangContext.Consumer>
    )
};

export default Title;
```

**4) useContext로 Context 접근하기**

**ReactHooks의 useContext 함수를 이용하면 좀 더 깔끔하게 Context에 저장되어 있는 전역 데이터를 접근할 수 있다. useContext 함수에 LangContext를 넘김으로써 사용자 언어 설정값을 읽는다. 단 이 방법은 함수 컴포넌트에서만 사용 가능하다.**

**[src]-[components]-[Button.js]**

```
import { useContext } from "react";
import LangContext from "../contexts/LangContext";

const Button = ({onToggleLang}) => {
    const lang= useContext(LangContext);
    return <button onClick={onToggleLang}>{lang}</button>
};

export default Button;
```

**5) contextType으로 context 접근하기**

**Message 컴포넌트와 같이 클래스로 구현된 컴포넌트의 경우에는 contextType을 사용해서 Context에 저장되어 있는 전역 데이터에 접근할 수 있다. this.context를 조회했을 때 현재 Context의 value를 가리키게 된다. 단 이 방법은 클래스 컴포넌트에서만 사용 가능하다.**

**[src]-[components]-[Message.js]**

```
import { Component } from "react";
import LangContext from '../contexts/LangContext';

class  Message  extends Component {
    static contextType = LangContext;
    render(){
        const lang = this.context;
        if(lang === 'en'){
            return( <div><p>Context provides a way to ...</p></div> );
        }else {
            return( <div> <p>컨텍스트는 ... 제공합니다.</p></div> );
        }
    }
};

export default Message;
```

- **Context API 사용법 익히기**

**1) 새 Context 만들기 : src 디렉터리에 contexts 디렉터리를 만든 뒤 그 안에 color.js라는 파일을 만든다.**

**[src]-[contexts]-[Color.js]**

```
import { createContext } from "react";
const ColorContext = createContext({ color: 'black' });
export default ColorContext;
```

**2) Consumer 사용하기 : ColorBox라는 컴포넌트를 만들어서 ColorContext 안에 들어 있는 색상을 보여준다.**

**[src]-[components]-[ColorBox.js]**

```
import ColorContext from "../contexts/Color";

const ColorBox = () => {
    return(
        <ColorContext.Consumer>
            { value => (
                <div style = {{ width: '64px', height: '64px', background: value.color}}/>
            )}
        </ColorContext.Consumer>
    );
};

export default ColorBox;
```

3) Provider 사용하기

기존에 createContext 함수를 사용할 때는 파라미터로 Context의 기본값을 넣어 주었다. 이 기본값은 Provider를 사용하지 않을 때만 사용한다. 만약 Provider는 사용했는데 value를 명시하지 않았다면 이 기본값을 사용하지 않기 때문에 오류가 발생한다.

[src]-[App.js]

```
import ColorBox from "./components/ColorBox";
import ColorContext from "./contexts/Color";

const App = () => {
    return(
        <ColorContext.Provider value={{color: 'red'}}>
            <div>
                <ColorBox/>
            </div>
        </ColorContext.Provider>
    );
};

export default App;
```

- 동적 Context 사용하기

지금까지 배운 내용으로는 고정적인 값만 사용할 수 있었다. 이번에 Context의 값을 업데이트해야 하는 경우를 알아보겠다.

1) Context 파일 수정하기

Context value에는 무조건 상태 값만 있어야 하는 것은 아니다. 함수를 전달해 줄 수도 있다. 아래 예제에서는 ColorProvider라는 컴포넌트를 새로 작성해 주었다. 이 Provider의 value에는 상태로 state를 업데이트 함수는 actions로 묶어서 전달하고 있다. Context에서 값을 동적으로 사용할 때 반드시 묶어줄 필요는 없지만 이렇게 state와 actions 객체를 따로따로 분리해 주면 나중에 다른 컴포넌트에서 Context의 값을 사용할 때 편리하다.

추가로 createContext를 사용할 때는 기본값으로 사용할 객체도 수정했다. createContext의 기본값은 실제 Provider의 value에 넣는 객체의 형태와 일치시켜 주는 것이 좋다. 그렇게 하면 Context 코드를 볼 때 내부 값이 어떻게 구성되어 있는지 파악하기도 쉽고 실수로 Provider를 사용하지 않았을 때 리액트 애플리케이션에서 에러가 발생하지 않는다.

[src]-[contexts]-[Color.js]

```
const { createContext, useState } = require("react");

const ColorContext = createContext({
    state: { color: 'black', subColor: 'red'},

    actions: {
        setColor: () => {},
        setSubColor: () => {}
    }
});

const ColorProvider = ({ children }) => {
    const [color, setColor] = useState('black');
    const [subColor, setSubColor] = useState('red');

    const value = {
        state: { color, subColor },
        actions: { setColor, setSubColor }
    };

    return(
        <ColorContext.Provider value={value}>{children}</ColorContext.Provider>
    );
};

//const ColorConsumer = ColorContext.Consumer와 같은 의미
const { Consumer: ColorConsumer} = ColorContext;

//ColorProvider와 ColorConsumer 내보내기
export { ColorProvider, ColorConsumer }

export default ColorContext;
```

### 2) 새로워진 Context를 프로젝트에 반영하기

[src]-[App.js]

```
import ColorBox from "./components/ColorBox";
import { ColorProvider } from "./contexts/Color";

const App = () => {
    return(
        <ColorProvider>
            <ColorBox/>
        </ColorProvider>
    );
};

export default App;
```

[src]-[components]-[ColorBox.js]

```
import { ColorConsumer } from "../contexts/Color";

const ColorBox = () => {
    return(
        <ColorConsumer>
            { value => (
                <div>
                    <div style = {{ width: '64px', height: '64px', background: value.state.color}}/>
                    <div style = {{ width: '32px', height: '32px', background: value.state.subColor}}/>
                </div>
            )}
        </ColorConsumer>
    );
};

export default ColorBox;
```

**위 코드에서 객체 비구조화 할당 문법을 사용하면 아래와 같이 value를 조회하는 것을 생략할 수 있다.**

[src]-[components]-[ColorBox.js]

```
<ColorConsumer>
    { ({ state }) => (
        <div>
            <div style = {{ width: '64px', height: '64px', background: state.color}}/>
            <div style = {{ width: '32px', height: '32px', background: state.subColor}}/>
        </div>
    )}
</ColorConsumer>
```

### 3) 색상 선택 컴포넌트 만들기

**이번에는 Context의 actions에 넣어 준 함수를 호출하는 컴포넌트를 아래와 같이 만들어 본다.**

[src]-[components]-[SelectColros.js]

```
const colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'];
const SelectColors = () => {
    return (
        <div>
            <h2>색상을 선택하세요.</h2>
            <div style={{ display: 'flex'}}>
                {colors.map(color => (
                    <div key={color} style={{background: color, width: '24px', height:'24px', cursor:'pointer'}}/>
                ))}
            </div>
            <hr/>
        </div>
    );
};

export default SelectColors;
```

[src]-[components]-[App.js]

```
import ColorBox from "./components/ColorBox";
import SelectColors from "./components/SelectColors";
import { ColorProvider } from "./contexts/Color";

const App = () => {
    return(
        <ColorProvider>
            <SelectColors/>
            <ColorBox/>
        </ColorProvider>
    );
};

export default App;
```

SelectColor에서 마우스 왼쪽을 클릭하면 큰 정사각형의 색상을 변경하고 마우스 오른쪽을 클릭하면 작은 정사각형의 색상을 변경해 보자. 마우스 오른쪽 버튼 클릭 이벤트는 onContextMenu를 사용하면 된다. 오른쪽 클릭 시 원래 브라우저 메뉴가 나타나지만 e.preventDefault()를 호출하면 메뉴가 뜨지 않는다.

[src]-[components]-[SelectColros.js]

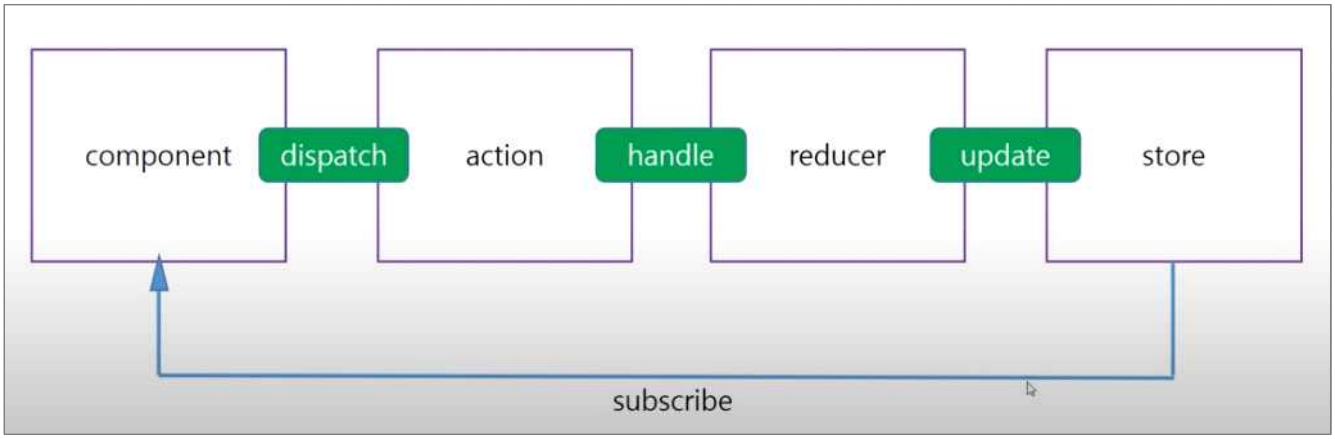
```
import { ColorConsumer } from "../contexts/Color";

const colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'];
const SelectColors = () => {
    return (
        <div>
            <h2>색상을 선택하세요.</h2>
            <ColorConsumer>
                {({ actions }) => (
                    <div style={{ display: 'flex'}}>
                        {colors.map(color => (
                            <div key={color} style={{background: color, width: '24px', height:'24px', cursor:'pointer'}}
                                onClick = {()=> actions.setColor(color)}
                                onContextMenu={ e => {
                                    e.preventDefault();  //마우스 오른쪽 버튼 클릭시 메뉴가 뜨는 것을 무시함
                                    actions.setSubColor(color);
                                }}
                            />
                        ))}
                    </div>
                )}
            </ColorConsumer>
            <hr/>
        </div>
    );
};

export default SelectColors;
```

- 실행 결과


기존에는 컴포넌트 간에 상태를 교류해야 할 때 모조건 부모 -> 자식 흐름으로 props를 통해 전달해 주었다. 이제는 Context API를 통해 쉽게 상태를 교류할 수 있게 되었다. 전역적으로 사용되는 상태가 있고 컴포넌트의 개수가 많은 상황이라면 Context API를 사용을 권한다.

# 12장 리덕스을 사용하여 리액트 애플리케이션 상태 관리하기 (ex04)

**리덕스는 가장 많이 사용하는 리액트 상태 관리 라이브러리이다. 리덕스를 사용하면 컴포넌트의 상태 업데이트 관련 로직을 다른 파일로 분리시켜 더욱 효율적으로 관리할 수 있다. 또한, 컴포넌트끼리 똑같은 상태를 공유해야 할 때도 여러 컴포넌트를 거치지 않고 손쉽게 상태 값을 전달하거나 업데이트할 수 있다. 때문에 전역 상태 관리할 때 굉장히 효과적이다.**




- **개념 미리 정리하기**

**1) 액션 : 상태에 어떠한 변화가 필요하면 액션(action)이란 것이 발생한다. 이는 하나의 객체로 표현된다. 액션 객체는 type 필드를 반드시 가지고 있어야 한다. 이 값을 액션의 이름이라고 생각하면 된다. 그리고 그 외의 값들은 나중에 업데이트를 할 때 참고해야 할 값이며 작성자 마음대로 넣을 수 있다. 액션 객체는 아래와 같은 형식으로 이루어져 있다.**

```
{
    type: 'ADD_TODO',
    data: { id: 1, text: '리덕스 배우기' }
}
```

**2) 액션 생성 함수: 액션 생성 함수는 액션 객체를 만들어 주는 함수이다. 어떤 변화를 일으켜야 할 때마다 액션 객체를 만들어야 하는데 매번 액션 객체를 직접 작성하기 번거로울 수 있고 만드는 과정에서 실수로 정보를 놓칠 수도 있다. 이러한 일을 방지하기 위해 이를 함수로 만들어 관리한다.**

```
const addTodo = (data) => {
    type: 'ADD_TODO',
    data
}
```

**3) 리듀서: 리듀서(reducer)는 변화를 일으키는 함수이다. 액션을 만들어서 발생시키면 리듀서가 현재 상태와 전달받은 액션 객체를 파라미터로 받아 온다. 그리고 두 값을 참고하여 새로운 상태를 만들어서 반환한다.**

```
const initialState = {
    counter: 1
};

function reducer(state = initialState, action) {
    switch(action.type) {
    case INCREMENT:
        return {
            counter: state.counter + 1
        };
    default:
        return state;
    }
}
```

**4) 스토어 : 프로젝트에 리덕스를 적용하기 위해 스토어(store)를 만든다. 한 개의 프로젝트는 단 하나의 스토어만 가질 수 있다. 스토어 안에는 현재 애플리케이션 상태와 리듀서가 들어가 있으며 그 외에도 몇 가지 중요한 내장 함수를 지닌다.**

**5) 디스패치: 디스패치(dispatch)는 스토어의 내장 함수로 액션을 발생시키는 것'이라고 이해하면 된다. 이 함수는 dispatch(action)과 같은 형태로 액션 객체를 파라미터로 넣어서 호출한다. 이 함수가 호출되면 스토어는 리듀서 함수를 실행시켜서 새로운 상태를 만들어 준다.**

**6) 구독: 구독(subscribe)도 스토어의 내장 함수 중 하나이다. subscribe 함수 안에 리스너 함수를 파라미터로 넣어서 호출해 주면, 이 리스너 함수가 액션이 디스패치되어 상태가 업데이트될 때마다 호출한다.**

```
const listener = () => {
    console.log('상태가 업데이트됨');
}

const unsubscribe = store.subscribe(listener);
unsubscribe();  //추후 구독을 비활성화할 때 함수를 호출
```

- react 컴포넌트 생성 단축키 : [Extensions] 에서 React snippets 2.0.1 를 검색해서 install한다. (단축키: rc)

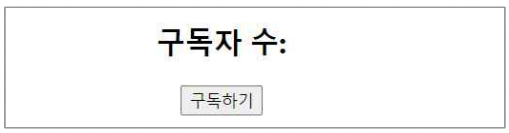
- 생성한 프로젝트 디렉터리에 yarn 명령어를 사용하여 리덕스와 react-redux 라이브러리를 설치한다.

```
yarn add redux react-redux
```

- Subscribers 컴포넌트 만들기

[src]-[components]-[Subscribers.js]

```
const Subscribers = (props) => {
    return (
        <div className="items">
            <h2>구독자 수: { props.count }</h2>
            <button>구독하기</button>
        </div>
    );
}
```




[src]-[App.css]

```
.items {
    border-bottom: #333 1px solid;
    margin-bottom: 1rem;
    padding-bottom: 1rem;
}
```

[src]-[App.js]

```
import './App.css';
import Subscribers from './components/Subscribers';

function App() {
    return (
        <div className="App"><Subscribers/></div>
    );
}
export default App;
```

- 리덕스(redux)관련 코드 작성하기 : 리덕스 관련 코드를 작성한다. 리덕스를 사용할 때는 액션 타입, 액션 생성 함수, 리듀서 코드를 작성해야한다. 이 코드들을 각각 다른 파일에 작성하는 방법도 있고, 기능별로 묶어서 파일 하나에 작성하는 방법도 있다. 여기서는 기능별로 파일 하나로 작성하는 Ducks 패턴 방식을 사용하였다.

1) 액션 타입 정의하기 : module 디렉터리를 생성하고 그 안에 subscribers.js 파일을 다음과 같이 작성한다.

[src]-[modules]-[subscribers.js]

```
//액션 타입 정의하기
const ADD_SUBSCRIBER = 'subscribers/ADD_SUBSCRIBER';  //액션 이름 충돌을 피하기 위해 문자열 안에 모듈 이름을 넣는다.
const REMOVE_SUBSCRIBER = 'subscribers/REMOVE_SUBSCRIBER';
```

2) 액션 생성 함수 만들기 : 액션 타입을 정의한 다음에는 액션 생성 함수를 만들어 주어야 한다.

[src]-[modules]-[subscribers.js]

```
//액션 생성 함수 만들기
export const  addSubscriber = () => {  //다른 파일에서 사용하기위해 export 키워드를 넣어준다.
    return {
        type: ADD_SUBSCRIBER
    }
};

export const removeSubscriber = () => {
    return {
        type: REMOVE_SUBSCRIBER
    }
}
```

3) 초기 상태 및 리듀서 함수 만들기 : subscribers 모듈의 초기 상태와 리듀서 함수를 만들어 준다.

[src]-[modules]-[subscribers.js]

```
//리듀서 작성하기
const initialState = {  //count 초기값을 설정해 준다.
    count: 370
}

const subscribers = (state=initialState, acition) => {
    switch(acition.type){
        case ADD_SUBSCRIBER:
            return {
                ...state,
                count: state.count + 1
            }
        case REMOVE_SUBSCRIBER:
            return {
                ...state,
                count: state.count - 1
            }
        default:
            return state
    }
}

export default subscribers;  //export는 여러 개를 내보낼 수 있지만 export default는 단 한 개만 보낼 수 있다.
```

4) 루트 리듀서 만들기 : 스토어를 만들 때 리듀서를 하나만 사용해야 하므로 여러개의 리듀서를 하나로 합쳐주어야 한다.

[src]-[module]-[rootReducer.js]

```
import { combineReducers } from "redux";
import subscribers from './subscribers';

const rootReducer = combineReducers({
    subscribers
});

export default rootReducer;
```

- 리액트 애플리케이션에 리덕스 적용하기

이제 드디어 리액트 애플리케이션에 리덕스를 적용할 차례이다. 스토어를 만들고 리액트 애플리케이션에 리덕스를 적용하는 작업을 한다.

1) 스토어 만들기 : 가장 먼저 [src]-[index.js] 파일에 스토어를 생성한다.

[src]-[index.js]

```
...
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './module/rootReducer';

const store = createStore(rootReducer);
...
```

2) Provider 컴포넌트를 사용하여 프로젝트에 리덕스 적용하기 : 스토어를 사용할 수 있도록 App 컴포넌트를 Provider로 감싸준다.

[src]-[index.js]

```
...
ReactDOM.render(
    <Provider store={store}>
        <App />
    </Provider>,
    document.getElementById('root')
);
...
```

- **SubScribers 컴포넌트 수정하기**

**컴포넌트에서 리덕스 스토어에 접근하여 원하는 상태를 받아오고, 또 액션도 디스패치해줄 차례이다.**

[src]-[components]-[subScriberReducer.js]

```
import React from 'react';
import { addSubscriber } from '../modules/subscribers';
import { connect } from 'react-redux';

const Subscribers = ({count, addSubscriber}) => {
    return (
        <div className="items">
            <h2>구독자 수: {count}</h2>
            <button onClick={addSubScriber}>구독하기</button>
        </div>
    );
};

const mapStateToProps = ({subscribers}) => {  //state를 파라미터로 받아 오며 이 값은 현재 스토어가 지니고 있는 상태를 의미한다.
    return {
        count: subscribers.count
    }
}

//const mapDispatchToProps = (dispath) => {
//    return { addSubscriber : () => dispath(addSubscriber()) }
//}

const mapDispatchToProps = {  //store의 내장 함수 dispatch를 파라미터로 받아 온다.
    addSubscriber
}
export default connect(mapStateToProps, mapDispatchToProps)(Subscribers);
```


- **Display 컴포넌트 만들기 : Display 컴포넌트를 만들어 증가된 구독자 수를 출력해 본다.**

[src]-[components]-[Display.js]

```
import React from 'react';
import { connect } from 'react-redux';

const Display = ({count}) => {
    return (
        <div className="items">
            <p>구독자 수는 {count}명 입니다.</p>
        </div>
    );
};

const mapStateToProps = ({subscribers}) => {
    return {
        count : subscribers.count
    }
}
export default connect(mapStateToProps)(Display);
```


- **조회수 컴포넌트 만들기**

[src]-[components]-[Views.js]

```
import React from 'react';

const Views = ({number, addView}) => {
    return (
        <div className="items">
            <h2>조회 수: {number}</h2>
            <button>조회하기</button>
        </div>
    );
};
export defualt Views
```

[src]-[modules]-[view.js]

```
//액션 타입 정의하기
const ADD_VIEW = 'views/ADD_VIEW';

//액션 생성 함수 만들기
export const addView = () => {
    return {
        type: ADD_VIEW
    }
}

//리듀서 작성하기
const initialState = {
    count: 0
}

export const views = (state=initialState, action) => {
    switch(action.type){
        case ADD_VIEW:
            return {
                ...state,
                count: state.count + 1
            }
        default:
            return state;
    }
}

export default views;
```

[src]-[components]-[Views.js] 수정

```
import React from 'react';
import { addView } from '../modules/views';
import { connect } from 'react-redux';

const Views = ({count, addView}) => {
    return (
        <div className="items">
            <h2>조회 수: {count}</h2>
            <button onClick={addView}>조회하기</button>
        </div>
    );
};
const mapStateToProps = ({views}) => {
    return {
        count: views.number
    }
}
const mapDispatchToProps = {
    addView
}
export default connect(mapStateToProps, mapDispatchToProps)(Views);
```


[src]-[App.js]

```
..
import Views from './components/Views';

function App() {
    return (
        <div className="App">
            <Subscribers/>
            <Views/>
            <Display/>
        </div>
    );
}

export default App;
```

- **텍스트 상자에 입력된 값으로 조회 수 증가시키기**

[src]-[modules]-[views.js]

```
//액션 타입 정의하기
const ADD_VIEW = 'views/ADD_VIEW';

//액션 생성 함수 만들기
export const addView = (number) => {
    return {
        type: ADD_VIEW,
        number: parseInt(number) //숫자형으로 변환
    }
}

//리듀서 작성하기
const initialState = {
    count: 0
}

export const views = (state=initialState, action) => {
    switch(action.type){
        case ADD_VIEW:
            return {
                ...state,
                count: state.count + action.number
            }
        default:
            return state;
    }
}
export default views;
```

[src]-[modules]-[rootReducer.js]

```
import { combineReducers } from "redux";
import subscribers from './subscribers';
import views from './views';

const rootReducer = combineReducers({
    subscribers,
    views,
});
export default rootReducer;
```

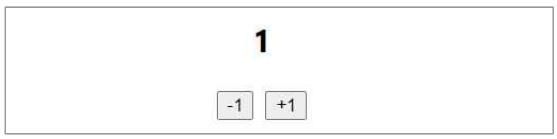
[src]-[components]-[Views.js]

```
import React, { useState } from 'react';
import { addView } from '../modules/views';
import { connect } from 'react-redux';

const Views = ({count, addView}) => {
    const [number, setNumber] = useState(1)
    return (
        <div className="items">
            <h2>조회 수: {count}</h2>
            <input type="text" value={number} onChange={(e)=>setNumber(e.target.value)}/>
            <button onClick={()=>addView(number)}>조회하기</button>
        </div>
    );
};

const mapStateToProps = ({views}) => {
    return {
        count: views.count
    }
}

const mapDispatchToProps = {
    addView
}
export default connect(mapStateToProps, mapDispatchToProps)(Views);
```

- **Counter 컴포넌트 만들기**

[src]-[components]-[Counter.js]

```
import React from 'react';

const Counter = ( props ) => {
    return (
        <div className="items">
            <h2>{props.number}</h2>
            <button>-1</button>
            <button>+1</button>
        </div>
    );
};
```




[src]-[modules]-[counter.js]

```
//액션 타입 정의하기
const INCREASE = 'counter/INCREASE';
const DECREASE = 'counter/DECREASE';

//액션 생성 함수 만들기
export const  increase = () => {
    return {
        type: INCREASE
    }
};

export const decrease = () => {
    return {
        type: DECREASE
    }
}

//리듀서 작성하기
const initialState = {
    number: 0
}

const counter = (state=initialState, action) => {
    switch(action.type){
        case INCREASE:
            return {
                ...state,
                number: state.number + 1
            }
        case DECREASE:
            return {
                ...state,
                number: state.number - 1
            }→
        default:
            return state;
    }
}

export default counter;
```

[src]-[App.js]

```
..
import counter from './counter';

const rootReducer = combineReducers({
    subscribers,
    views,
    counter
});

export default rootReducer;
```

[src]-[components]-[Counter.js]

```
import React from 'react';
import {connect} from 'react-redux';
import { increase, decrease } from '../modules/counter';

const Counter = ({ number, increase, decrease }) => {
    return (
        <div className="items">
            <h2>{number}</h2>
            <button onClick={decrease}>-1</button>
            <button onClick={increase}>+1</button>
        </div>
    );
};

const mapStateToProps = ({counter}) => {
    return {
        number: counter.number + 1
    }
}

const mapDispatchProps = {
    increase,
    decrease
}
export default connect(mapStateToProps, mapDispatchProps)(Counter);
```

[src]-[App.js]

```
...
import Counter from './components/Counter';

function App() {
    return (
        <div className="App">
            <Subscribers/>
            <Views/>
            <Display/>
            <Counter/>
        </div>
    );
}
export default App;
```

- **할 일 목록 컴포넌트 만들기**

[src]-[components]-[Todos.js]

```
import React from 'react';

const TodoItem = () => {
    return (
        <div>
            <input type="checkbox"/>
            <span>예제 텍스트</span>
            <button>삭제</button>
        </div>
    )
}

const Todos = () => {
    return (
        <div className="items">
            <form><input/><button type="submit">등록</button></form>
            <div><TodoItem/><TodoItem/><TodoItem/><TodoItem/><TodoItem/></div>
        </div>
    );
};
export default Todos;
```

[src]-[modules]-[todos.js]

```
//액션 타입 정의하기
const CHANGE_INPUT = 'todos/CHANGE_INPUT';
const INSERT = 'todos/INSERT';
const TOGGLE = 'todos/TOGGLE';
const REMOVE = 'todos/REMOVE';

//액션 생성 함수 만들기
export const changeInput = (input) => {
    return {
        type: CHANGE_INPUT,
        input: input
    }
};

let id = 3;
export const insert = (text) => {
    return {
        type: INSERT,
        todo: { id: id++, text, done: false }
    }
};
export const toggle = (id) => {
    return {
        type: TOGGLE,
        id: id
    }
};
export const remove = (id) => {
    return {
        type: REMOVE,
        id: id
    }
};

//리듀서 작성하기
const initialState = {
    input: '',
    todos: [
        { id: 1, text:'리덕스 기초 배우기', done: true },
        { id: 2, text:'리액트와 리덕스 사용하기', done: false }
    ]
};

const todos = (state=initialState, action) => {
    switch(action.type){
        case CHANGE_INPUT:
            return {
                ...state,
                input: action.input
            }
        case INSERT:
            return {
                ...state,
                todos: state.todos.concat(action.todo)
            }
        case TOGGLE:
            return {
                ...state,
                todos: state.todos.map(todo => todo.id === action.id ? {...todo, done: !todo.done} : todo)
            }
        case REMOVE:
            return {
                ...state,
                todos: state.todos.filter(todo => todo.id !== action.id)
            }
        default:
            return state
    }
}

export default todos;
```

[src]-[modules]-[rootReducer.js]

```
...
import todos from './todos';

const rootReducer = combineReducers({
    subscribers,
    views,
    counter,
    todos
});

export default rootReducer;
```

[src]-[components]-[Todos.js]

```
import React from 'react';
import { connect } from 'react-redux';
import { changeInput, insert, toggle, remove } from '../modules/todos';

const TodoItem = ({ todo, onRemove, onToggle }) => {
    return (
        <div>
            <input type="checkbox" checked={todo.done} readOnly={true} onChange={onToggle}/>
            <span style={{textDecoration: todo.done ? 'line-through': 'none'}}>{todo.text}</span>
            <button onClick={onRemove}>삭제</button>
        </div>
    )
}

const Todos = ({ todos, input, changeInput, insert, remove, toggle }) => {
    const onSubmit = (e) => {
        e.preventDefault();
        insert(input);
        changeInput('');
    }

    const onRemove = (id) => {
        if(!window.confirm(`${id}를(을) 삭제하실래요?`) return;
        remove(id);
    }
    return (
        <div className="items">
             <h2>할일 목록</h2>
            <form onSubmit={onSubmit}>
                <input value={input} onChange={(e)=>changeInput(e.target.value)}/>
                <button type="submit">등록</button>
            </form>
            <div>
                { todos.map(todo =>
                    <TodoItem key={todo.id} todo={todo} onRemove={()=>onRemove(todo.id)} onToggle={()=>toggle(todo.id)}/>
                )}
            </div>
        </div>
    );
};

const mapStateToProps = ({ todos }) =>{
    return {
        input: todos.input,
        todos: todos.todos
    }
}

const mapDispatchProps = {
    changeInput,
    insert,
    toggle,
    remove
}

export default connect(mapStateToProps, mapDispatchProps)(Todos);
```

# 13. 리덕스 미들웨어를 통한 비동기 작업 관리

웹 애플리케이션에서 API 서버를 연동할 때는 API 요청에 대한 상태도 잘 관리해야 한다. 요청이 성공하면 서버에서 받아 온 응답에 대한 상태를 관리하고 요청이 실패하면 서버에서 반환한 에러에 대한 상태를 관리해야 한다. 리액트 프로젝트에서 리덕스를 사용하고 있으며 이러한 비동기 작업을 관리해야 한다면 '미들웨어'를 사용하여 매우 효율적이고 편하게 상태 관리를 할 수 있다.

## • 미들웨어란?

리덕스 미들웨어는 액션을 디스패치했을 때 리듀서에서 이를 처리하기에 앞서 사전에 지정된 작업들을 실행한다. 미들웨어는 액션과 리듀서 사이의 중간자라고 볼 수 있다. 리듀서가 액션을 처리하기 전에 미들웨어가 할 수 있는 작업은 여러 가지가 있다. 전달받은 액션을 단순히 콘솔에 기록하거나 전달받은 액션 정보를 기반으로 액션을 아예 취소하거나 다른 종류의 액션을 추가로 디스패치할 수도 있다.


### • redux-logger 사용하기

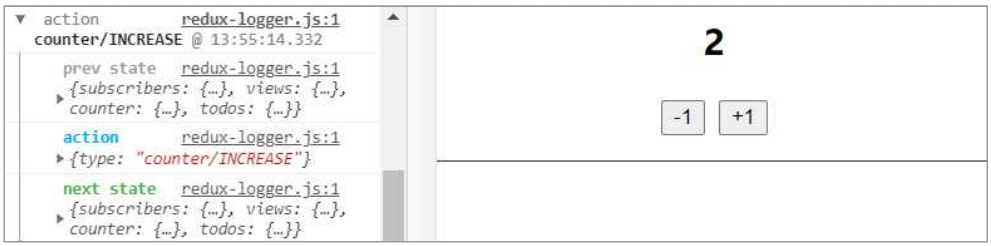
오픈 소스 커뮤니티에 올라와 있는 redux-logger 미들웨어를 설치하고 사용해 보자. 브라우저 콘솔에 형식이 깔끔하게 출력된다.

```
yarn add redux-logger
```

[src]-[index.js]

```
...
import { createStore, applyMiddleware } from 'redux';
import { createLogger } from 'redux-logger';

const logger = createLogger();
const store = createStore(rootReducer, applyMiddleware(logger));
...
```

콘솔에 색상과 액션 디스패치 시간이 나타난다. 미들웨어를 사용할 때는 완성된 미들웨어를 라이브러리로 설치해서 사용하는 경우가 많다.




## • 비동기 작업을 처리하는 미들웨어 사용

이제 오픈 소스 커뮤니티에 공개된 미들웨어를 사용하여 리덕스를 사용하고 있는 프로젝트에서 비동기 작업을 더욱 효율적으로 관리해 보겠다.

### • redux-turnk

Tunk는 특정 작업을 나중에 할 수 있도록 미루기 위해 함수 형태로 감싸는 것을 의미한다. 예를 들어 주어진 파라미터에 1을 더하는 함수를 만들고 싶다면 아래와 같이 작성할 것이다. 이 코드를 실행하면 addOne을 호출했을 때 바로 1 + 1이 연산된다.

```
const addOne = x => x + 1;
addOne(1);  //2
```

그런데 위의 연산을 나중에 하도록 미루고 싶다면 다음과 같이 구현할 수 있다.

```
const addOne = x => x + 1;
function addOneThunk(x) {  //const addOneThunk = x => () => addOne(x);  화살표 함수로 사용한 경우
    const thnk = () => anddOne(x);
    return thnk;
}
const fn = addOneThunk(1)
setTimeout( ()=> {
    const value=fn();  //fn이 실행되는 시점에 연산
    console.log(value);
}, 1000);
```

redux-thunk 라이브러리를 사용하면 thunk 함수를 만들어서 디스패치할 수 있다. 그러면 리덕스 미들웨어가 그 함수를 전달 받아 store의 dispatch와 getState를 파라미터로 넣어서 호출해 준다. 아래는 redux-thunk에사 사용할 수 있는 예시 thunk함수 이다.

```
const sampleThunk = () => (dispatch, getState) => {
    // 현재 상태를 참조할 수 있고,
    // 새 액션을 디스패치할 수도 있다.
}
```

- 미들웨어 적용하기 : redux-thunk 미들웨어를 설치하고 프로젝트에 적용해 본다.

```
yarn add redux-thunk
```

[src]-[index.js]

```
...
import { createStore, applyMiddleware } from 'redux';
import { createLogger } from 'redux-logger';
import ReduxTunk from 'redux-thunk';

const logger = createLogger();

const store = createStore(rootReducer, applyMiddleware(logger, ReduxTunk));
...
```

- Thunk 생성 함수 만들기 : redux-thunk는 액션 생성 함수에서 일반 액션 객체를 반환하는 대신에 함수를 반환한다.

[src]-[modules]-[counter.js]

```
//액션 생성 함수 만들기
export const increaseAsync = () => dispatch => {
    setTimeout(()=>{dispatch(increase())}, 1000);
}

export const decreaseAsync = () => dispatch => {
    setTimeout(()=>{dispatch(decrease())}, 1000);
}
```

개발자 도구를 열어서 버튼을 클릭한 후 숫자가 1초 뒤에 변경되는지 확인해 본다.

[src]-[components]-[Counter.js]

```
import React from 'react';
import {connect} from 'react-redux';
import { increaseAsync, decreaseAsync } from '../modules/counter';

const Counter = ({number, increaseAsync, decreaseAsync}) => {
    return (
        <div className="items">
            <h2>{number}</h2>
            <button onClick={ decreaseAsync }>-1</button>
            <button onClick={ increaseAsync }>+1</button>
        </div>
    );
};

const mapStateToProps = ({counter}) => {
    return {
        number: counter.number + 1
    }
}

const mapDispatchProps = {
    increaseAsync,
    decreaseAsync
}

export default connect(mapStateToProps, mapDispatchProps)(Counter);
```

- **웹 요청 비동기 작업 처리하기 (comments)**

**thunk의 속성을 활용하여 웹 요청을 연습하기 위해 JSONplaceholder에서 제공되는 가짜 API를 사용하겠다. 사용할 API는 다음과 같다.**

```
https://jsonplaceholder.typicode.com/comments
```

**1) 액션 타입, 액션 생성 함수, 리듀서 정의 한다.**

[src]-[module]-[comments.js]

```
//액션 타입 정의하기
const FETCH_COMMENTS_REQUEST = 'comments/FETCH_COMMENTS_REQUEST';
const FETCH_COMMENTS_SUCCESS = 'comments/FETCH_COMMENTS_SUCCESS';
const FETCH_COMMENTS_FAILURE = 'comments/FETCH_COMMENTS_FAILURE';

//액션 생성 함수 만들기
export const fetchComments = () => {
    return (dispatch) => {
        fetch("https://jsonplaceholder.typicode.com/comments")
        .then(response => response.json())
        .then(comments => console.log(comments))
        .catch(error => console.log(error))
    }
}

//리듀서 작성하기
const initialState = {
    items: []
}
export const comments = (state=initialState, action) => {
    switch(action.type){
        default:
            return state;
    }
}
export default comments;
```

[src]-[modules]-[rootReducer.js]

```
...
import comments from './comments'

const rootReducer = combineReducers({
    comments
});
export default rootReducer;
```

[src]-[components]-[Comments.js]

```
import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { fetchComments } from '../modules/comments';

const Comments = ({ fetchComments }) => {
    useEffect(() => {
        fetchComments()
    }, []);

    return (
        <div className="items"></div>
    );
};
const mapStateToProps = ({ comments }) => {
    return {
        comments: comments.items
    }
}
const mapDispatchToProps = {
    fetchComments
}
export default connect(mapStateToProps, mapDispatchToProps)(Comments);
```

2) Comments 내용 출력을 위한 프로그램 수정 한다.


**[src]-[module]-[comments.js]**

```
..
const initialState = {
    items: [], loading: false, error: null
}

export const fetchComments = () => {
    return (dispatch) => {
        dispatch(fetchCommentsRequest());
        fetch("https://jsonplaceholder.typicode.com/comments")
            .then(response => response.json())
            .then(items => dispatch(fetchCommentSuccess(items)))
            .catch(error => dispatch(fetchCommentsFailure(error)));
    }
}

const fetchCommentSuccess = (items) => {
    return { type: FETCH_COMMENTS_SUCCESS, payload: items }
}

const fetchCommentsFailure = (error) => {
    return { type: FETCH_COMMENTS_FAILURE, payload: error }
}

const fetchCommentsRequest = () => {
    return { type: FETCH_COMMENTS_REQUEST }
}

export const comments = (state=initialState, action) => {
    switch(action.type){
        case FETCH_COMMENTS_REQUEST:
            return {
                ...state,
                loading: true
            }
        case FETCH_COMMENTS_SUCCESS:
            return {
                ...state,
                items: action.payload,
                loading: false
            }
        case FETCH_COMMENTS_FAILURE:
            return {
                ...state,
                error: action.payload,
                loading: false
            }
        default:
            return state;
    }
}
export default comments;
```

[src]-[components]-[Comments.js]

```
import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { fetchComments } from '../modules/comments';

const Comments = ({ fetchComments, loading, items }) => {
    useEffect(() => {
        fetchComments()
    }, []);

    const commentsItems = loading ? (<div>로딩중...</div>) : (
        items.map(item => (
            <div key={item.id}>
                <h3>{item.name}</h3>
                <p>{item.email}</p>
                <p>{item.body}</p>
            </div>
        ))
    );

    return (
        <div className="comments">
            {commentsItems}
        </div>
    );
};

const mapStateToProps = ({comments}) => {
    return {
        items: comments.items
    }
}

const mapDispatchToProps = {
    fetchComments
}

export default connect(mapStateToProps, mapDispatchToProps)(Comments);
```

3) Comments에 스타일 적용한다.

[src]-[App.css]

```
.comments {
    padding:10px;
    display: grid;
    grid-template-columns: repeat(3, 1fr);  /* 1fr 비율을 같게 */
    grid-gap: 1rem;
}

.comments > div {
    border: 1px solid #333;
}
```

- 웹 요청 비동기 작업 처리하기 (포스트 읽기)

API를 호출할 때는 주로 Promise 기반 웹 클라이언트인 axios를 사용하므로 해당 라이브러리를 아래와 같이 설치한다.

```
yarn add axios
```

특정 포스트 읽기를 위한 JSONplaceholder의 API는 아래와 같다.

```
https://jsonplaceholder.typicode.com/posts/:id
```

### 1) 액션 타입, 액션 생성 함수, 리듀서 정의한다.

[src]-[modules]-[posts.js]

```
import axios from 'axios';

//액션 타입 정의하기
const GET_POST = 'posts/GET_POST';
const GET_POST_SUCCESS = 'posts/GET_POST_SUCCESS';
const GET_POST_FAILURE = 'posts/GET_POST_FILURE';

//액션 생성 함수 만들기
export const fetchPosts = (id) =>async dispatch => {
    dispatch(getPost());
    try {
        const response = await axios.get('https://jsonplaceholder.typicode.com/posts/' + id);
        dispatch(getPostSuccess(response));
    }catch(error) {
        dispatch(getPostFailure(error));
    }
}

const getPost = () => {
    return { type: GET_POST }
}

const getPostSuccess = (response) => {
    return {
        type: GET_POST_SUCCESS,
        payload: response.data
    }
}

const getPostFailure = (error) => {
    return {
        type: GET_POST_FAILURE,
        payload: error
    }
}

//리듀서 작성하기
const initialState = { tem : null

const posts = (state=initialState, action) => {
    switch(action.type){
        case GET_POST:
            return {
                ...state,
            }
        case GET_POST_SUCCESS:
            return {
                ...state,
                item: action.payload,
                loading: false
            }
        case GET_POST_FAILURE:
            return {
                ...state,
                error: action.error,
                loading: false
            }
        default:
            return state;
    }
}

export default posts;
```

2) rootReducer에 posts 리듀서를 등록한다.

[src]-[modules]-[rootReducer.js]

```
...
import posts from './posts';

const rootReducer = combineReducers({
    posts
});

export default rootReducer;
```

3) Posts 컴포넌트를 작성한다.


[src]-[components]-[Posts.js]

```
import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { fetchPosts } from '../modules/posts';

const Posts = ({fetchPosts, loading, item}) => {
    useEffect(() => {
        fetchPosts(1)
    },[]);

    return (
        <div>
            <h1>포스트</h1>
            { loading && '로딩중' }
            { !loading && item && (
                <div>
                    <h2>{item.title}</h2>
                    <h3>{item.body}</h3>
                </div>
            ) }
        </div>
    );
};

const mapStateToProps = ({posts}) => {
    return {
        item: posts.item
    }
}

const mapDispatchToProps = {
    fetchPosts: (id) => fetchPosts(id)
}

export default connect(mapStateToProps, mapDispatchToProps)(Posts);
```

- 웹 요청 비동기 작업 처리하기 (모든 사용자 정보 불러오기)

모든 사용자 정보 불러오기를 위한 JSONplaceholder의 API는 아래와 같다.

```
https://jsonplaceholder.typicode.com/users
```

1) 액션 타입, 액션 생성 함수, 리듀서 정의한다.

[src]-[modules]-[users.js]

```
import axios from 'axios';

//액션 타입 정의하기
const GET_USERS = 'users/GET_USERS';
const GET_USERS_SUCCESS = 'users/GET_USERS_SUCCESS';
const GET_USERS_FAILURE = 'users/GET_USERS_FILURE';

//액션 생성 함수 만들기
export const fetchUsers = () =>async dispatch => {
    dispatch(getUsers());
    try {
        const response = await axios.get('https://jsonplaceholder.typicode.com/users/');
        dispatch(getUsersSuccess(response));
    }catch(error) {
        dispatch(getUsersFailure(error));
    }
}

const getUsers = () => {
    return {
        type: GET_USERS
    }
}

const getUsersSuccess = (response) => {
    return {
        type: GET_USERS_SUCCESS,
        payload: response.data
    }
}

const getUsersFailure = (error) => {
    return {
        type: GET_USERS_FAILURE,
        payload: error
    }
}

//리듀서 작성하기
const initialState = {
    items : []
}

export const users = (state=initialState, action) => {
    switch(action.type){
        case GET_USERS:
            return {
                ...state,
            }
        case GET_USERS_SUCCESS:
            return {
                ...state,
                items: action.payload,
                loading: false
            }
        case GET_USERS_FAILURE:
            return {
                ...state,
                error: action.error,
                loading: false
            }
        default:
            return state;
    }
}

export default users;
```

2) rootReducer에 posts 리듀서를 등록한다.

[src]-[modules]-[rootReducer.js]

```
...
import posts from './users';

const rootReducer = combineReducers({
    posts,
    users
});

export default rootReducer;
```

3) Users 컴포넌트를 아래와 같이 작성한다.


[src]-[components]-[Users.js]

```
import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { fetchUsers } from '../modules/users';

const Users = ({fetchUsers, loading, items}) => {
    useEffect(()=>{
        fetchUsers()
    }, []);

    return (
        <div className="items">
            <h2>사용자 목록</h2>
            { loading && '로딩중...' }
            { !loading && items && (
                <ul>
                    {items.map(item => (
                        <li key={item.id}>
                            {item.username} ({item.email})
                        </li>
                    ))}
                </ul>
            )}
        </div>
    );
};

const mapStateToProps = ({users}) => {
    return {
        items: users.items
    }
}

const mapDispatchToProps = {
    fetchUsers
}

export default connect(mapStateToProps, mapDispatchToProps)(Users);
```

# 13장 백엔드 프로그래밍

**우리는 서버를 만들어 데이터를 여러 사람과 공유한다. 그런데 서버에 데이터를 담을 때는 아래와 같은 사항들을 고려해야한다. 아래 내용들을 고려해서 로직을 만드는 것을 서버 프로그래밍 또는 백엔드(back-end) 프로그래밍 이라고 한다.**

- **데이터를 등록할 때 사용자 인증 정보가 필요한지 고려한다.**
- **등록할 데이터를 어떻게 검증할지 고려한다.**
- **데이터 종류가 다양하다면 어떻게 구분할지 고려해야 한다.**
- **데이터를 조회할 때도 어떤 종류의 데이터를 몇 개씩 어떻게 보여줄지 고려해야 한다.**
-

**백엔드 프로그래밍은 여러 가지 환경으로 진행할 수 있다. 즉 언어에 구애받지 않기 때문에 PHP, 파이썬, Golan, 자바, 자바스크립트, 루비 등과 같은 다양한 언어로 구현할 수 있다. 그중 자바스크립트로 서버를 구현할 수 있는 Node.js를 사용해 보겠다.**

## • Node.js Express 서버 개발 환경 구축하기

**1) 새로운 리액트 프로젝트를 생성한다.**

```
yarn create react-app ex05
```

**2) 파일관리자에서 [ex05]폴더에서 [client] 폴더를 생성한 후 [ex05] 프로젝트의 모든 파일을 [client] 폴더로 이동시킨다.**

**3) [client] 폴더 안에 [.gitignore] 파일을 [ex05] 프로젝트 안에 복사한다.**

**4) [ex05] 폴더에서 웹서버 구동을 위한 nodemon을 설치하면 [ex05]-[node_modules] 폴더가 생성된다.**

```
npm install nodemon
```

**5) [ex05] 프로젝트의 폴더에 [package.json] 파일을 생성한 후 아래 내용을 작성한다.**

```
[ex05]-[package.json]
```

```
{
    "name": "management",
    "version": "1.0.0",
    "private": true,
    "scripts": {
        "client": "cd client && yarn start",
        "server": "nodemon server.js",
        "dev": "concurrently --kill-others-on-fail \"yarn server\" \"yarn client\""
    }
}
```

**6) [ex05]폴더에서 웹서버를 구축하기 위한 express와 서버와 클라이언트를 동시에 실행하기 위한 concurrently를 아래와 같이 설치한다.**

```
npm install express concurrently
```

- **라이브러리를 설치한 후 [package.json] 파일의 dependencies에 라이브러리들이 추가되었는지 확인하다.**

```
[ex05]-[package.json]
```

```
{
    "name": "management",
    "version": "1.0.0",
    "private": true,
    "scripts": {
        "client": "cd client && yarn start",
        "server": "nodemon server.js",
        "dev": "concurrently --kill-others-on-fail \"yarn server\" \"yarn client\""
    },
    "dependencies": {
        "concurrently": "^6.2.0",
        "express": "^4.17.1"
    }
}
```

7) 웹서버 구축을 위한 server.js 파일을 아래와 같이 작성한다.

[ex05]-[server.js]

```
const express = require('express');
const app = express();
const port = process.env.PORT || 5000;
app.use(express.json());

app.listen(port, () => {
    console.log(`server is listening at localhost:${port}`);
});
app.get('/', (req, res) => {
    res.send({success: true})
});
```

8) 웹서버가 잘 작동하는지 nodemon을 이용하여 실행한 후 브라우저 주소에 http://localhost:5000/을 입력해 본다.

```
nodemon server.js
```

9) 웹서버와 클라이언트 서버를 동시에 실행을 위해 아래 명령어를 입력해 본다.

```
yarn dev
```

- **고객 관리 시스템 프로그래밍**


- 고객 컴포넌트 구조화

[ex05]-[client]-[src]-[components]-[Customer.js]

```
const CustomperProfile = (props) => {
    const {id, name, image } = props;
    return (
        <div><img src={image} alt="profile"/><h2>{id}({name})</h2></div>
    );
}
const CustomerInfo = (props) => {
    const {birthday, gender, job} = props;
    return (
        <div><p>{birthday}</p><p>{gender}</p><p>{job}</p></div>
    );
}
const Customer = (props) => {
    const {id, name, image, birthday, gender, job} = props.customer;
    return (
        <div>
            <CustomperProfile id={id} image={image} name={name}/>
            <CustomerInfo birthday={birthday} gender={gender} job={job}/>
        </div>
    );
};
export default Customer;
```

[ex05]-[client]-[src]-[components]-[Customers.js]

```
import Customer from './components/Customer';

const customers = [
    { id: 1, image:'https://placeimg.com/64/64/1', name:'홍길동', birthday:'961222', gender:'남자', job:'대학생' },
    { id: 2, image:'https://placeimg.com/64/64/2', name:'심청이', birthday:'960506', gender:'여자', job:'회사원' },
    { id: 3, image:'https://placeimg.com/64/64/3', name:'강감찬', birthday:'861202', gender:'남자', job:'프로그래머' },
];

function Customers() {
    return (
        <div>
            { customers.map(customer => (
                <Customer key={customer.id} customer={customer}/>
            )) }
        </div>
    );
}

export default App;
```

- **Material UI를 적용하려 고객 테이블 디자인하기**

| 번호 | 이미지 | 이름 | 생년월일 | 성별 | 직업 |
|---|---|---|---|---|---|
| 1 |  | 홍길동 | 961222 | 남자 | 대학생 |
| 2 |  | 심청이 | 960506 | 여자 | 회사원 |
| 3 |  | 강감찬 | 861202 | 남자 | 프로그래머 |

**https://mui.com 설치 매뉴얼을 참고하여 아래와 같이 라이브러리를 설치한다.**

```
npm install @mui/material @emotion/react @emotion/styled
```

[ex05]-[client]-[src]-[components]-[Customer.js]

```
import React from 'react';
import { TableCell, TableRow } from '@mui/material';

const Customer = ({customer}) => {
    const { id, name, image, birthday, gender, job } = customer;
    return (
        <TableRow>
            <TableCell>{id}</TableCell>
            <TableCell><img src={image} alt="profile"/></TableCell>
            <TableCell>{name}</TableCell>
            <TableCell>{birthday}</TableCell>
            <TableCell>{gender}</TableCell>
            <TableCell>{job}</TableCell>
        </TableRow>
    );
};
export default Customer;
```

[ex05]-[client]-[src]-[components]-[Customers.js]

```
import React, {useState, useEffect} from 'react';
import {Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow} from '@mui/material';
import Customer from './Customer';

const Customers = () => {
    return (
        <div>
            <TableContainer component={Paper}>
                <Table sx={{minWidth: 650}} aria-label="simple table">
                    <TableHead>
                        <TableRow>
                            <TableCell>번호</TableCell>
                            <TableCell>이미지</TableCell>
                            <TableCell>이름</TableCell>
                            <TableCell>생년월일</TableCell>
                            <TableCell>성별</TableCell>
                            <TableCell>직업</TableCell>
                        </TableRow>
                    </TableHead>
                    <TableBody>
                        {customers ? customers.map(customer=>(
                            <Customer key={customer.id} customer={customer}/>
                        )) : ''}
                    </TableBody>
                </Table>
            </TableContainer>
        </div>
    );
};

export default Customers;
```

- Node.js Express에서 REST API 구축하기

1) App.js에 있던 customers 데이터를 server.js로 이동시킨다.

[ex05]-[server.js]

```
app.get('/api/customers', (req, res) => {
    res.send([
        { id: 1, image:'https://placeimg.com/64/64/1', name:'홍길동', birthday:'961222', gender:'남자', job:'대학생' },
        { id: 2, image:'https://placeimg.com/64/64/2', name:'심청이', birthday:'960506', gender:'여자', job:'회사원' },
        { id: 3, image:'https://placeimg.com/64/64/3', name:'강감찬', birthday:'861202', gender:'남자', job:'프로그래머' },
    ]);
});
```

2) http://localhost:5000/api/customers에 접속하여 출력되는 JSON 데이터를 아래 주소에서 올바른 데이터 형식인지 테스트 해본다.

https://jsonlint.com

3) client 폴더의 package.json에 아래와 같이 proxy 서버를 설정한다. 프록시 서버는 클라이언트가 자신을 통해서 다른 네트워크 서비스에 간접적으로 접속할 수 있게 해 주는 컴퓨터 시스템이나 응용 프로그램을 가리키다.

[ex05]-[client]-[package.json]

```
{
    ...
    "proxy": "http://localhost:5000/"
    ...
}
```

4) 프로그래바 UI를 사용하기 위해 아래 주소로 이동하여 예제를 참조한다.

https://mui.com/components/progress/

[ex05]-[client]-[src]-[Customers.js]

```
import React, {useState, useEffect} from 'react';
import {Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow} from '@mui/material';
import Customer from './Customer';
import CircularProgress from '@mui/material/CircularProgress';
import Box from '@mui/material/Box';

const Customers = () => {
    const [customers, setCustomers] = useState('');
    const [loading, setLoading] = useState(false);

    const callAPI = () => {
        setLoading(true);
        fetch('/api/customer/s')
            .then(res => res.json())
            .then(json => {
                setCustomers(json);
                setLoading(false);
                console.log(JSON.stringify(json, null, 2));
        });
    }

    useEffect(() => {
        callAPI();
    },[]);

    return (
        <div>
            <TableContainer component={Paper}>
                <Table sx={{minWidth: 650}} aria-label="simple table">
                    <TableHead>
                        <TableRow>
                            <TableCell>번호</TableCell>
                            <TableCell>이미지</TableCell>
                            <TableCell>이름</TableCell>
                            <TableCell>생년월일</TableCell>
                            <TableCell>성별</TableCell>
                            <TableCell>직업</TableCell>
                        </TableRow>
                    </TableHead>
                    <TableBody>
                        {customers ? customers.map(customer=>(
                            <Customer key={customer.id} customer={customer}/>
                        )) : ''}
                    </TableBody>
                </Table>
            </TableContainer>
            { loading ?
                <Box sx={{textAlign: 'center', paddingTop:'50px'}}>
                    <CircularProgress/>
                </Box>:'' }
        </div>
    );
};

export default Customers;
```

```
useEffect(() =>{  //2초 후에 callAPI() 실행
    setLoading(true);
    setTimeout(()=> {
        callAPI()
            .then(res => {
                setCustomers(res)
                setLoading(false);
            })
            .catch(err => {
                console.log(err);
                setLoading(false)
            });
    }, 2000);
},[] );
```

- **AWS RDS 서비스를 이용하여 MySQL DB 구축하기**

1. https://aws.amazon.com/ko로 접속한 후 회원가입하고 새 계정을 생성한다.
2. 서비스 찾기에서 cost explorer에서 비용을 확인 후 사용하지 않을 경우 서비스를 종료해 준다.
3. 서비스 찾기에서 RDS 검색 후 MySQL 서비스를 신청한다.
4. https://www.heidisql.com/download.php에서 MySQL 클라이언트 프로그램을 설치한다.

- **고객(Customer) DB 테이블 구축 및 Express와 연동하기**

**1) 데이터베이스 테이블 생성 및 Sample Data 입력한다.**

데이터베이스 및 테이블 생성

```
create database management;
use management;
create table customers(
    id int primary key auto_increment,
    image varchar(1024),
    name varchar(64),
    birthday varchar(64),
    gender varchar(64),
    job varchar(64)
);
insert into customers(image,name,birthday,gender,job) values('https://placeimg.com/64/64/1','홍길동','961222','남자','대학생');
insert into customers(image,name,birthday,gender,job) values('https://placeimg.com/64/64/2','심청이','960509','여자','회사원');
insert into customers(image,name,birthday,gender,job) values('https://placeimg.com/64/64/3','강감찬','861202','남자','의사');
```

**2) 서버 폴더에서 mysql 라이브러리를 아래와 같이 설치한다.**

```
yarn add mysql
```

**3) database.json 파일을 생성한다.**

[ex05]-[database.json]

```
{
    "host": "localhost",
    "user": "root",
    "password": "1234",
    "port": "3306",
    "database": "management"
}
```

**4) server.js 파일을 아래와 Database에 접속하는 모듈을 추가한다.**

[ex05]-[server.js]

```
const fs = require('fs')
const db = fs.readFileSync('./database.json');
const conf = JSON.parse(db);
const mysql = require('mysql');

const connection = mysql.createConnection({
    host: conf.host,
    user: conf.user,
    password: conf.password,
    port: conf.port,
    database: conf.database
});

//connection.connect();

app.get('/api/customers', (req, res) => {
    connection.query('select * from customers',
        (err, rows, fields) => { res.send(rows); }
    );
});
```

- **고객 추가 양식(Form) 구현 및 이벤트 핸들링**

**1) 서버와의 비동기 통신을 위한 axios 라이브러리를 설치한다.**

```
yarn add axios
```

**2) CustomerAdd.js 추가 양식 파일을 작성한다.**

[ex05]-[client]-[src]-[customers]-[CutomerAdd.js]

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

const CustomerAdd = () => {
    const [form, setForm] = useState({
        file: null, fileName: '', userName: '', birthday: '', gender: '', job: '',
    });

    const { file, fileName, userName, birthday, gender, job } = form;

    const onChange = (e) => {
        const nextForm = {
            ...form,
            [e.target.name]: e.target.value
        };
        setForm(nextForm);
    }

    const onFileChange = (e) => {
        const nextForm = {
            ...form,
            file: e.target.files[0],
            fileName: e.target.value
        }
        setForm(nextForm);
    }

    const addCustomer = () => {
        const url = '/api/customers';
        const formData = new FormData();
        formData.append('image', file);
        formData.append('name', userName);
        formData.append('birthday', birthday);
        formData.append('gender', gender);
        formData.append('job', job);
        const config = {
            Headers: { 'content-type': 'multipart/form-data' }
        };

        return post(url, formData, config);
    }

    const onSubmit = (e) => {
        e.preventDefault();
        addCustomer()
            .then(res => { console.log(res.data) });
    }

    return (
        <form onSubmit={ onSubmit }>
            프로필 이미지: <input type="file" name="file" value={fileName} onChange={onFileChange}/><br/>
            이름: <input type="text" name="userName" value={userName} onChange={onChange}/><br/>
            생년월일: <input type="text" name="birthday" value={birthday} onChange={onChange}/><br/>
            성별: <input type="text" name="gender" value={gender} onChange={onChange}/><br/>
            직업: <input type="text" name="job" value={job} onChange={onChange}/><br/>
            <button type="submit">추가하기</button>
        </form>
    );
};

export default CustomerAdd;
```

```
//이미지 미리보기
const [imgSrc, setImgSrc] = useState('https://placeimg.com/150/150/1')
const onFileChange = (e) => {
    var reader=new FileReader();
    reader.onload = e => {
        setImgSrc(e.target.result);
    }
    reader.readAsDataURL(e.target.files[0]);
}
const refImage = useRef(null);

return (
    <div>
        <img src={imgSrc} alt="temp"/>
        <input type="file" onChange={ onFileChange } ref={ refImage }/>
        <button onClick={ ()=>refImage.current.click() }>파일선택하기</button>
    </div>
);
```

3) Customers.js 파일에 CustomerAdd.js 컴포넌트를 추가한다. ( F12-[Network]-[customers]-[Headers]에서 Form Data를 확인한다. )

[ex05]-[client]-[src]-[components]-[Customers.js]

```
..
import CustomerAdd from './components/CustomerAdd';

function Customers() {
    ...
    return (
        <div>
            ...
            <CustomerAdd/>
        </div>
    );
}
export Customers;
```


- Node.js Express에서 파일 업로드 요청 처리 및 DB에 데이터 삽입

1) 파일 업로드 처리 작업을 위해 multer 라이브러리를 설치한다.

```
yarn add multer  (npm install --save multer)
```

2) 고객 데이터를 테이블에 추가하고 파일업로드를 위한 서버프로그램을 추가한다.

[ex05]-[server.js]

```
...
//파일 업로드
const multer = require('multer');
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, './upload');  //업로드 패스를 지정  './public/upload' 물리적 path 지정
    },
    filename: (req, file, cb) => {
        cb(null, `${Date.now()}_${file.originalname}`);  //새로운 파일이름을 업로드날짜_오리지널이름으로 지정
    }
});
const upload = multer({ storage: storage });
app.use('/image', express.static('./upload'));  //upload 폴더를 접근할 때 image 폴더로 접근

//고객 테이블에 등록
app.post('/api/customers', upload.single('image'), (req, res) => {
    let sql = 'insert into customers(image,name,birthday,gender,job) values(?,?,?,?,?)';
    let image = '/image/' + req.file.filename;  //'/upload/' + req.file.filename
    let name = req.body.name;
    let birthday = req.body.birthday;
    let gender = req.body.gender;
    let job = req.body.job;
    let params = [image, name, birthday, gender, job];
    connection.query(sql, params, (err, rows, fields) => {
        res.sendStatus(200);
    });
});
```

3) [ex05] 프로젝트 폴더 아래 [upload] 폴더를 새로 생성한 후 CustomerAdd.js에 '새로고침'을 위한 내용을 추가한다.

[ex05]-[client]-[components]-[CustomerAdd.js]

```
const onSubmit = (e) => {
    e.preventDefault();
    addCustomer()
        .then(res => { console.log(res.data) });

    setForm({ file: null, fileName: '', userName: '', birthday: '', gender: '', job: '' });
    window.location.reload();
}
```

- **부모 컴포넌트의 상태 변경을 통한 고객정보 갱신**

**1) 고객 목록에서 'axios'를 이용해 출력하도록 변경하고 callAPI()를 CustomerAdd 컴포넌트로 넘겨준다.**

[ex05]-[client]-[src]-[customers]-[Customers.js]

```
...
import axios from 'axios';
...

function Customers() {
    const [customers, setCustomers] = useState(null);
    const [loading, setLoading] = useState(false);

    useEffect(() => {
        callAPI();
    },[] );

    const callAPI = async () => {
        setLoading(true);
        try{
            const response = await axios.get('/api/customers');
            setCustomers(response.data);
        }catch(e){
            console.log(e)
        }
        setLoading(false);
    }

    return (
        <div>
            ...
            <CustomerAdd callAPI={callAPI}/>
        </div>
    );
}

export default Customers;
```

```
useEffect(() =>{
    setLoading(true);
    callAPI()
        .then(res => {
            setCustomers(res)
            setLoading(false);
        })
        .catch(err => {
            console.log(err);
            setLoading(false)
      });
}, []);

const callAPI = async() => {
    const response = await fetch('/api/customers');
    const body = await response.json();
    return body;
}
```

**2) CustomerAdd 컴포넌트에서 callAPI() 함수를 props로 받아 submit() 함수에서 실행한다.**

[ex05]-[client]-[src]-[components]-[CustomerAdd.js]

```
...

const CustomerAdd = ({ callAPI }) => {
    ...
    const onSubmit = (e) => {
        e.preventDefault();

        addCustomer().then(res => {
            console.log(res.data) ;  //console.log(res.status) 상태코드값
            setForm({
                file: null,
                fileName: '',
                userName: '',
                birthday: '',
                gender: '',
                job: ''});
            callAPI();
        });
    }

    return (
         <form onSubmit={ onSubmit }>
            ...
        </form>
    );
};

export default CustomerAdd;
```

- **고객(Customer)정보 삭제 기능 구현하기**

| 번호 | 이미지 | 이름 | 생년월일 | 성별 | 직업 | 삭제 |
|---|---|---|---|---|---|---|
| 2 |  | 심청이 | 960509 | 여자 | 회사원 | 삭제 |
| 3 |  | 강감찬 | 861202 | 남자 | 프로그래머 | 삭제 |

**1) customer 테이블에 두 개의 칼럼(등록한 날짜와 삭제유무)을 추가한다.**

**customers 테이블 수정**

```
alter table customers add createDate datetime;
alter table customers add isDeleted int;
update customers set createDate = now() where id > 0;
update customers set isDeleted = 0 where id > 0;
```

**2) id를 받아 테이블에서 데이터를 삭제하고 callAPI로 고객 목록을 다시 출력하는 CustomerDelete 컴포넌트를 작성한다.**

**[ex05]-[client]-[src]-[components]-[CustomerDelete.js]**

```
const CustomerDelete = ({ callAPI, id }) => {
    const deleteCustomer = (id) => {
        const url = '/api/customers/' + id;
        fetch(url, { method: 'delete' });
        callAPI();
    }

    return ( <button onClick={()=>deleteCustomer(id)}>삭제</button> );
};
```

```
const onDelete = (id)=> {
    if(!window.confirm(`${id}를(을) 삭제하실래요?`)) return;
    post(`/customer/delete/${id}`).then(res=>{
        if(res.status ===200) {
            alert('고객등록삭제완료!');
            callAPI();
        }
    });
}
```

**3) props(callAPI() 함수, id)를 속성으로 하는 CustomerDelete 컴포넌트를 Customer 컴포넌트에 추가한다.**

**[ex05]-[client]-[src]-[components]-[Customer.js]**

```
import CustomerDelete from './CustomerDelete';
const Customer = (props) => {
    const {id, name, image, birthday, gender, job} = props.customer;
    return (
            <TableRow>
                ...
                <TableCell><CustomerDelete callAPI={ props.callAPI } id={ id }/></TableCell>
            </TableRow>
    );
};
```

**4) Customer 컴포넌트에 props로 callAPI() 함수를 넘겨주는 App 컴포넌트를 수정한다.**

**[ex05]-[client]-[src]-[components]-[Customers.js]**

```
<TableHead>
    <TableRow>
        ...
        <TableCell>삭제</TableCell>
    </TableRow>
</TableHead>
<TableBody>
    { customers ? customers.map(customer => (
        <Customer key={customer.id} customer={customer} callAPI={callAPI}/>
    )) : ''}
</TableBody>
```

5) server.js 파일에서 고객목록출력, 등록 sql문을 수정하고 고객 삭제 모듈을 추가한다.

[ex05]-[server.js]

```
//고객 테이블에 등록
app.post('/api/customers', upload.single('image'), (req, res) => {
    let sql = 'insert into customers(image,name,birthday,gender,job,isDelete,createDate) values(?,?,?,?,?,0,now())';
    ...
});

//고객목록출력
app.get('/api/customers', (req, res) => {
    connection.query('select * from customers where isDeleted=0',
        (err, rows, fields) => { res.send(rows); }
    );
});

//고객 삭제
app.delete('/api/customers/:id', (req, res) => {
    let sql = 'update customers set isDeleted=1 where id=?';
    let params = [req.params.id];
    connection.query(sql, params, (err, rows, fields) => {
       res.send(rows);
    });
});
```

- Material UI 모달(Modal) 디자인 구현하기

1) 아래 url 사이트로 이동하여 모달(Model) 디자인을 참조한다.

https://mui.com/components/dialogs/

2) CustomerDelete 컴포넌트에 Confirm 모달창을 추가한다.

[ex05]-[client]-[src]-[components]-[CustomerDelete.js]

```
import { Button, Dialog, DialogActions, DialogContent, DialogTitle, Typography } from "@material-ui/core";
import { useState } from 'react';

const CustomerDelete = ({ callAPI, id }) => {
    const [open, setOpen] =  useState(false);
    const onClickOpen = () => {
        setOpen(true);
    }
    const onClickClose = () => {
        setOpen(false);
    }
    const deleteCustomer = (id) => {
        const url = '/api/customers/' + id;
        fetch(url, { method: 'delete' });
        callAPI();
    }
    return (
        <div>
            <Button variant="contained" color="error" onClick={onClickOpen}>삭제</Button>
            <Dialog open={open}>
                <DialogTitle>삭제경고</DialogTitle>
                <DialogContent gutterBottom>
                    <Typography>선택한 고객 정보가 삭제됩니다.</Typography>
                </DialogContent>
                <DialogActions>
                    <Button variant="contained" onClick={()=>deleteCustomer(id)}>삭제</Button>
                    <Button variant="outlined" onClick={onClickClose}>닫기</Button>
                </DialogActions>
            </Dialog>
        </div>
    );
};

export default CustomerDelete;
```

3) CustomerAdd 컴포넌트를 아래와 같이 수정한다.

[ex05]-[client]-[src]-[components]-[CustomerAdd.js]

```
import React, { useState } from 'react';
import {post} from 'axios';
import {Button, Dialog, DialogActions, DialogContent, DialogTitle, TextField} from '@mui/material'

const CustomerAdd = ({callAPI}) => {
    ...
    const [open, setOpen] = useState(false);
    const onClickOpen = () => { setOpen(true); }
    const onClickClose = () => {
        onReset();
        setOpen(false);
    }
    const onReset = () => {
        setForm({
            file: null,
            fileName: '',
            userName: '',
            birthday: '',
            gender: '',
            job: ''
        });
    }

    const onSubmit = (e) => {
        e.preventDefault();
        if(!window.confirm("고객을 등록하실래요?")) return;
        addCustomer().then(res=>{
            console.log(JSON.stringify(res,null,2));
            if(res.status===200) {
                onReset();
                setOpen(false);
                alert("고객등록성공!");
                callAPI();
            }
        });
    }

    return (
        <div>
            <Button variant="contained" onClick={onClickOpen}>고객 추가하기</Button>
            <Dialog open={open}>
                <DialogTitle>고객 추가</DialogTitle>
                <DialogContent>
                    <input type="file" onChange={onFileChange}style={{display:'none'}} id="file-button" accept="image/*"/>
                    <label htmlFor="file-button">
                        <Button variant="contained" name="file" component="span">
                            {fileName ==='' ? '프로필 이미지 선택' : fileName}
                        </Button>
                    </label>
                    <br/>
                    <TextField label="성명" name="userName" value={userName} onChange={onChange} variant="standard"/>
                    <br/>
                    <TextField label="생년월일" name="birthday" value={birthday} onChange={onChange} variant="standard"/>
                    <br/>
                    <TextField label="성별" name="gender" value={gender} onChange={onChange} variant="standard"/>
                    <br/>
                    <TextField label="직업" name="job" value={job} onChange={onChange} variant="standard"/>
                    <br/>
                </DialogContent>
                <DialogActions>
                    <Button variant="contained" onClick={onSubmit}>추가</Button>
                    <Button variant="outlined" onClick={onClickClose}>닫기</Button>
                </DialogActions>
            </Dialog>
        </div>
    );
};
export default CustomerAdd;
```

- AppBar 및 웹 폰트를 적용하여 디자인 개편하기

1) 아래 url 사이트로 이동하여 'App Bar with search field' 디자인을 참조한다.

```
https://mui.com/  //Search 상자에서 App Bar를 검색한다.
```

2) AppBar 디자인에 필요한 아이콘 라이브러리를 client 폴더에 설치한다.

```
cd client
yarn add @mui/icons-material
```

3) 'App Bar with search filed' 컴포넌트를 복사해서 새로 생성한다.

[ex05]-[client]-[src]-[components]-[SearchAppBar.js]

```
import * as React from 'react';
import { styled, alpha } from '@mui/material/styles';
import AppBar from '@mui/material/AppBar';
import Box from '@mui/material/Box';
import Toolbar from '@mui/material/Toolbar';
import IconButton from '@mui/material/IconButton';
import Typography from '@mui/material/Typography';
import InputBase from '@mui/material/InputBase';
import MenuIcon from '@mui/icons-material/Menu';
import SearchIcon from '@mui/icons-material/Search';

const Search = styled('div')(({ theme }) => ({
    ...
}));

const SearchIconWrapper = styled('div')(({ theme }) => ({
    ...
}));

const StyledInputBase = styled(InputBase)(({ theme }) => ({
    ...
}));

export default function SearchAppBar() {
    return (
        <Box sx={{ flexGrow: 1 }}>
            <AppBar position="static">
                <Toolbar>
                    <IconButton size="large" edge="start" color="inherit" aria-label="open drawer" sx={{ mr: 2 }}>
                        <MenuIcon />
                    </IconButton>
                    <Typography variant="h6" noWrap component="div" sx={{flexGrow:1, display:{xs:'none', sm:'block'}}}>
                        고객 관리 시스템
                    </Typography>
                    <Search>
                        <SearchIconWrapper>
                            <SearchIcon />
                        </SearchIconWrapper>
                        <StyledInputBase placeholder="Search…" inputProps={{ 'aria-label': 'search' }}/>
                    </Search>
                </Toolbar>
            </AppBar>
        </Box>
    );
}
```

4) Customers 컴포넌트에 SearchAppBar 컴포넌트를 추가하고 스타일을 변경한다.

[ex05]-[client]-[src]-[components]-[Customers.js]

```
import React, {useState, useEffect} from 'react';
import { Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow} from '@mui/material';
import Customer from './Customer';
import CircularProgress from '@mui/material/CircularProgress';
import Box from '@mui/material/Box';
import CustomerAdd from './CustomerAdd';
import SearchAppBar from './SearchAppBar';

const Customers = () => {
    const [customers, setCustomers] = useState('');
    const [loading, setLoading] = useState(false);

    const callAPI = () => {
        setLoading(true);
        fetch('/customer/list')
            .then(res => res.json())
            .then(json => {
                setCustomers(json);
                setLoading(false);
                console.log(JSON.stringify(json, null, 2));
        });
    }

    useEffect(() => {
        callAPI();
    },[]);

    const headers =['번호','프로필이미지','이름','생년월일','성별', '직업','설정']

    return (
        <div>
            <SearchAppBar/>
            <div style={{margin:'20px', textAlign:'center'}}>
                <CustomerAdd callAPI={callAPI}/>
            </div>
            <TableContainer component={Paper}>
                <Table sx={{minWidth: 650}} aria-label="simple table">
                    <TableHead>
                        <TableRow>
                            { headers.map(header=>
                                <TableCell key={header}>{header}</TableCell>)
                            }
                        </TableRow>
                    </TableHead>
                    <TableBody>
                        { customers ? customers.map(customer=>(
                            <Customer key={customer.id} customer={customer} callAPI={callAPI}/>
                        )) : '' }
                    </TableBody>
                </Table>
            </TableContainer>
            { loading ? <Box sx={{textAlign: 'center', paddingTop:'50px'}}><CircularProgress/></Box>:'' }
        </div>
    );
};

export default Customers;
```

- **고객(Customer) 검색 기능 구현하기**

[ex05]-[client]-[src]-[components]-[Customers.js]

```
const Customers = () => {
    ...
    const callAPI = async (searchKey) => {
        setLoading(true);
        try{
            const response = await axios.get('/api/customers?searchKey=' + searchKey);
            setCustomers(response.data);
        }catch(e){
            console.log(e)
        }
        setLoading(false);
    }

    useEffect(() => {
        callAPI('');
    },[]);

    return (
        <div>
            <SearchAppBar callAPI={ callAPI }/>
            ...
        </div>
  );
}
```

[ex05]-[client]-[src]-[components]-[SearchAppBar.js]

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

export default function SearchAppBar({callAPI}) {
    const [searchKey, setSearchKey] = useState('');
    const onChange = (e) => {
        setSearchKey(e.target.value);
    }
    const onKeyPress = (e) => {
        if(e.key === 'Enter') {
            callAPI(searchKey);
        }
    }

    return (
        <Box sx={{ flexGrow: 1 }}>
            <AppBar position="static">
                <Toolbar>
                    <IconButton size="large" edge="start" color="inherit" aria-label="open drawer" sx={{ mr: 2 }}>
                        <MenuIcon />
                    </IconButton>
                    <Typography variant="h6" noWrap component="div" sx={{flexGrow:1, display:{xs: 'none', sm: 'block' } }}>
                        고객 관리 시스템
                    </Typography>
                    <Search>
                        <SearchIconWrapper>
                            <SearchIcon />
                        </SearchIconWrapper>
                        <StyledInputBase placeholder="Search…" inputProps={{ 'aria-label': 'search' }}
                                                                onChange={onChange} onKeyPress={onKeyPress}/>
                    </Search>
                </Toolbar>
            </AppBar>
        </Box>
    );
}
```

[ex05]-[server.js]

```
//고객목록출력
app.get('/api/customers', (req, res) => {
    let params = '%' + req.query.searchKey + '%';

    connection.query(
        'select * from customers where isDeleted=0 and name like ?',
        params,
        (err, rows, fields) => {
            res.send(rows);
        }
    );
});
```

[ex05]-[client]-[src]-[components]-[CustomerAdd.js]

```
const onSubmit = (e) => {
    e.preventDefault();
    if(!window.confirm("고객을 등록하실래요?")) return;
    addCustomer().then(res=>{
        console.log(JSON.stringify(res,null,2));
        if(res.status===200) {
            onReset();
            setOpen(false);
            alert("고객등록성공!");
            callAPI('');
        }
    });
}
```

[ex05]-[client]-[src]-[components]-[CustomerDelete.js]

```
const deleteCustomer = (id) => {
    const url = '/api/customers/' + id;
    fetch(url, { method: 'delete' });
    callAPI('');
}
```

- 웹 폰트 적용하기

1) 아래 url로 접속하여 원하는 웹 폰트를 다운로드 받은 후 폰트들을 [ex05]-[client]-[src]-[fonts] 폴더에 저장한다.

```
https://fonts.google.com/
```

2) index.css 파일에 아래 내용을 추가한다.

[ex05]-[client]-[index.css]

```
@font-face {
    font-family: 'Noto Sans KR Bold';
    src: url("./fonts/NotoSansKR-Black.otf")
}

@font-face {
    font-family: 'Noto Sans KR Medium';
    src: url("./fonts/NotoSansKR-Medium.otf")
}

@font-face {
    font-family: 'Noto Sans KR Thin';
    src: url("./fonts/NotoSansKR-Thin.otf")
}
```

# 14장 블로그 페이지 프로그래밍

## • 사용자, 데이터베이스, 테이블 생성

### 1) 사용자, 데이터베이스를 생성한 후 권한을 부여한다.

```
create user 'blog'@'localhost' identified by 'pass';
create database blogDB;
grant all privileges on blogDB.* to 'blog'@'localhost';
```

### 2) 데이터베이스에 테이블들을 생성한 후 샘플데이터를 입력한다.

User 테이블 생성 및 샘플 데이터 입력

```
create table tbl_user(
    userid varchar(20) not null primary key,
    username varchar(20),
    password varchar(100) not null
);

insert into tbl_user values('red', '홍길동', 'pass');
insert into tbl_user values('blue', '강감찬', 'pass');
insert into tbl_user values('yellow', '심청이', 'pass');
```

Post 테이블 생성 및 샘플 데이터 입력

```
create table tbl_post(
    id int auto_increment primary key,
    userid varchar(20) not null,
    title varchar(500) not null,
    body varchar(1000),
    createDate datetime default now(),
    foreign key(userid) references tbl_user(userid)
);

insert into tbl_post(userid, title, body)
values('red','왜 리액트인가?','한때 자바스크립트는 웹 브라우저에서 간단한 연산을 했지만 현재는 웹 애플리케이션에서 가장 핵심적인 역할을 한다.');

insert into tbl_post(userid, title, body)
values('blue','컴포넌트','컴포넌트를 선언하는 방식은 두 가지이다. 하나는 함수형 컴포넌트이고, 또 다른 하나는 클래스형 컴포넌트이다');

insert into tbl_post(userid, title, body)
values('yellow','배열 비구조화 할당','배열 안에 들어 있는 값을 쉽게 추출할 수 있도록 해 주는 문법이다.');
```

## • Node.js Express 서버 개발 환경 구축하기

### 1) 새로운 리액트 프로젝트를 생성한다.

```
yarn create react-app ex06
```

### 2) 파일관리자에서 [ex06]폴더에서 [client] 폴더를 생성한 후 [ex06] 프로젝트의 모든 파일을 [client] 폴더로 이동시킨다.

### 3) [client] 폴더 안에 [.gitignore] 파일을 [ex06] 프로젝트 안에 복사한다.

### 4) [ex06] 폴더에서 웹서버 구동을 위한 nodemon을 설치하면 [ex06]-[node_modules] 폴더가 생성된다.

```
npm install nodemon
```

5) [ex06] 프로젝트의 폴더에 [package.json] 파일을 생성한 후 아래 내용을 작성한다.

[ex06]-[package.json]

```
{
    "name": "blog",
    "version": "1.0.0",
    "private": true,
    "scripts": {
        "client": "cd client && yarn start",
        "server": "nodemon server.js",
        "dev": "concurrently --kill-others-on-fail \"yarn server\" \"yarn client\""
    }
}
```

6) [ex06]폴더에서 웹서버를 구축하기 위한 express와 서버와 클라이언트를 동시에 실행하기 위한 concurrently를 아래와 같이 설치한다.

```
npm install express concurrently
```

- 라이브러리를 설치한 후 [package.json] 파일의 dependencies에 라이브러리들이 추가되었는지 확인하다.

[ex06]-[package.json]

```
{
    "name": "management",
    "version": "1.0.0",
    "private": true,
    "scripts": {
        "client": "cd client && yarn start",
        "server": "nodemon server.js",
        "dev": "concurrently --kill-others-on-fail \"yarn server\" \"yarn client\""
    },
    "dependencies": {
        "concurrently": "^6.2.0",
        "express": "^4.17.1"
    }
}
```

7) 웹서버 구축을 위한 server.js 파일을 아래와 같이 작성한다.

[ex06]-[server.js]

```
const express = require('express');
const app = express();
const port = process.env.PORT || 5000;
app.use(express.json());

app.get('/', (req, res) => {
    res.send({success: true})
});

app.listen(port, () => {
    console.log(`server is listening at localhost:${port}`);
});
```

8) 웹서버가 잘 작동하는지 nodemon을 이용하여 실행한 후 브라우저 주소에 http://localhost:5000/을 입력해 본다.

```
nodemon server.js
```

9) 웹서버와 클라이언트 서버를 동시에 실행을 위해 아래 명령어를 입력해 본다.

```
yarn dev
```

- **Node.js Express에서 REST API 구축하기**

**1) 서버 폴더에서 mysql 라이브러리를 아래와 같이 설치한다.**

```
yarn add mysql
```

**2) database.json 파일을 생성한다.**

[ex06]-[database.json]

```
{ "host": "localhost", "user": "blog", "password": "pass", "port": "3306", "database": "blogdb" }
```

**3) server.js 파일을 아래와 Database에 접속하는 모듈을 추가한다.**

[ex06]-[server.js]

```
const express = require('express');
const app = express();
const port = process.env.PORT || 5000;
app.listen(port, () => {
    console.log(`server is listening at localhost:${port}`);
});
app.get('/', (req, res) => { res.send({success: true}) });

//데이터베이스 연결
const fs = require('fs')
const db = fs.readFileSync('./database.json');
const conf = JSON.parse(db);
const mysql = require('mysql');
exports.connection = mysql.createConnection({
    host: conf.host,
    user: conf.user,
    password: conf.password,
    port: conf.port,
    database: conf.database
});

app.use('/post', require('./routes/post'));
app.use('/auth', require('./routes/auth'));
```

**4) auth테이블에 대한 REST API를 작성한다.**

[ex06]-[routes]-[auth.js]

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

router.get('/:userid', (req, res) => {  //아이디,비밀번호 check
    var sql = 'select * from tbl_user where userid = ?'
    var userid = req.params.userid;
    db.connection.query(sql, [userid], (err, rows, fields) => { res.send(rows); } );
});

router.post('/register', (req, res) => {  //회원가입
    var userid = req.body.userid;
    var password = req.body.password;
    var sql = 'select * from tbl_user where userid = ?'
    db.connection.query(sql, [userid],
        (err, rows, fields) => {
            if(rows.length > 0) {
               res.send({'result': 1});
            }else{
                sql = 'insert into tbl_user(userid, password) values(?, ?)';
                db.connection.query(sql, [userid, password], (err, rows, fields) => { res.send({'result': 0}); });
            }
        }
    );
});

module.exports = router;
```

### 5) post 테이블에 대한 REST API를 작성한다. ( http://localhost:5000/post )

[ex06]-[routes]-[posts.js]

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

//게시글 목록
router.get('/', (req, res) => {
    var page = parseInt(req.query.page);
    var start = (page - 1) * 2;
    var sql = 'select *,date_format(createDate, "%Y-%m-%d %T") fmtCreateDate from tbl_post order by id desc  limit ?, 2';
    db.connection.query(sql, [start],
        (err, rows, fields) => { res.send(rows); }
    );
});

//게시글 등록
router.post('/write', (req, res) => {
    var userid = req.body.userid;
    var title = req.body.title;
    var body = req.body.body;

    var sql = 'insert into tbl_post(userid, title, body) values(?, ?, ?)';
    db.connection.query(sql, [userid, title, body],
        (err, rows, fields) => {
           res.send({ 'result': 1});
        }
    );
});

//게실글 읽기
router.get('/view/:postid', (req, res) => {
    var id = req.params.postid;
    var sql = 'select *, date_format(createDate, "%Y-%m-%d %T") fmtCreateDate from tbl_post where id=?';
    db.connection.query(sql, [id],
        (err, rows, fields) => {
           res.send(rows[0]);
        }
    );
});

//전체 게시글 수 읽기
router.get('/count', (req, res) => {
    var sql = 'select count(*) count from tbl_post';
    db.connection.query(sql,
        (err, rows, fields) => {
            res.send(rows[0]);
        });
});

//게시글 삭제
router.post('/delete', (req, res) => {
    var id = req.body.id;
    var sql = 'delete from tbl_post where id=?';
    db.connection.query(sql, [id],
        (err, rows, fields) => {
            res.send({ 'result': 1});
        }
    )
});

//게시글 수정
router.post('/update', (req, res) => {
    var id = req.body.id;
    var title = req.body.title;
    var body = req.body.body;
    var sql = 'update tbl_post set title=?, body=? where id=?';
    db.connection.query(sql, [title, body, id],
        (err, rows, fields) => {
            res.send({ 'result': 1});
        }
    )
});

module.exports = router;
```

- **프론트엔드 프로젝트**

- **[client] 폴더로 이동해 리액트 라우터 라이브러리를 아래와 같이 설치한다.**

```
yarn add react-router-dom@5.3.0
```

- **시작 프로그램 구현**

[ex06]-[client]-[src]-[index.js]

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>
  document.getElementById('root')
);

reportWebVitals();
```

[ex06]-[client]-[src]-[index.css]

```
body {
    margin: 0px;
    padding: 0px;

    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
                 'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue', sans-serif;
                 -webkit-font-smoothing: antialiased;
                 -moz-osx-font-smoothing: grayscale;
}

a {
    text-decoration: none;
}

code {
    font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New', monospace;
}
```

[ex06]-[client]-[src]-[App.js]

```
import { Route } from 'react-router-dom';
import PostListPage from './pages/PostListPage';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import WritePage from './pages/WritePage';
import PostPage from './pages/PostPage';
import './App.css';

function App() {
    return (
        <div className="App">
            <Route component={PostListPage} path='/' exact/>
            <Route component={LoginPage} path='/login'/>
            <Route component={RegisterPage} path='/register'/>
            <Route component={WritePage} path='/write'/>
            <Route component={PostPage} path='/view/:postid'/>
       </div>
    );
}

export default App;
```

[ex06]-[client]-[src]-[App.css]

```
.App {
    overflow: hidden;
    width: 1024px;
    margin: 0 auto;
}

@media screen and (max-width: 1024px) {
    .App {
        width: 768px;
    }
}

@media screen and (max-width: 768px) {
    .App {
        width: 100%;
    }
}

.grayButton {
    border: none;
    border-radius: 4px;
    font-size: 0.9rem;
    font-weight: bold;
    padding: 0.4rem 1rem;
    color: white;
    background: #343a40;
    cursor: pointer;
    margin-right: 1rem;
}

.grayButton:hover {
    background: #adb5db;
}

.skyButton {
    border: none;
    border-radius: 4px;
    font-size: 0.9rem;
    font-weight: bold;
    padding: 0.4rem 1rem;
    color: white;
    background: #22b8cf;
    cursor: pointer;
    margin-right: 1rem;
}

.skyButton:hover {
    background: #3bc9db;
}
```

- **회원인증 프로그램**

[ex06]-[client]-[src]-[pages]-[LoginPage.js]

```
import React from 'react';
import Auth from '../components/auth/Auth';

const LoginPage = () => {
    return (
        <div>
            <Auth/>
        </div>
    );
};

export default LoginPage;
```

[ex06]-[client]-[src]-[components]-[auth]-[Auth.js]

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import '../../css/Auth.css';
import axios from 'axios';
import { withRouter } from 'react-router';

const Auth = ({ history }) => {
    const [error, setError] = useState('');

    const [form, setForm] = useState({
        userid: '',
        password: '',
        username: ''
    });
    const {userid, password} = form;
    const onChange = (e) => {
        const nextForm = {
            ...form,
            [e.target.name]: e.target.value
        };
        setForm(nextForm);
    }

    const onSubmit = (e) => {
        e.preventDefault();
        setError('');
        if(userid === '' || password === '') {
            setError('빈 칸을 모두 입력하세요.');
            return;
        }
        const url = '/auth/' + userid;
        axios.get(url)
            .then(response => {
                if(response.data.length === 1) {
                    if(response.data[0].password === password){
                        sessionStorage.setItem('userid', response.data[0].userid);
                        history.push('/');
                    }else{
                        setError('비밀번호가 틀립니다.')
                    }
                }else{
                    setError('아이디가 없습니다.')
                }
        });
    };

    return (
        <div className="authBlock">
            <div className="whiteBox">
                <div className="logo-area">
                    <Link to='/'>REACTERS</Link>
                </div>
                <div className="authForm">
                    <h3>로그인</h3>
                    <form onSubmit={onSubmit}>
                        <input type="text" placeholder="아이디" name="userid" value={userid} onChange={onChange}/>
                        <input type="password" placeholder="비밀" name="password" value={password} onChange={onChange}/>
                        <button type="submit">로그인</button>
                    </form>
                    { error && <div className="errorMessage">{error}</div> }
                    <div className="footer">
                        <Link to="/register">회원가입</Link>
                    </div>
                </div>
            </div>
        </div>
    );
};

export default withRouter(Auth);
```

[ex06]-[client]-[css]-[Auth.css]

```
.authBlock {
    position: absolute;
    left: 0;
    right: 0;
    top: 0;
    bottom: 0;
    background: #e2e2e2;
    display:flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
.logo-area {
    font-size: 1.4rem;
    padding-bottom: 2rem;
    text-align: center;
    font-weight: bold;
    letter-spacing: 2px;
}

.whiteBox {
    width: 360px;
    box-shadow: 5px 5px 5px gray;
    border-radius: 10px;
    padding: 2rem;
    background: white;
}

.authForm  input {
    width: 100%;
    font-size: 1rem;
    border: none;
    padding-bottom: 0.5rem;
    outline: none;
    border-bottom: 1px solid #e2e2e2;
    margin-top: 1rem;
}
.authForm input:focus {
    border-bottom: 1px solid #495057;
}
.authForm button {
    width:100%;
    border: none;
    border-radius: 4px;
    font-size: 1rem;
    font-weight: bold;
    background: #22b8cf;
    color: white;
    margin-top: 1rem;
    padding-top: 0.7rem;
    padding-bottom: 0.7rem;
    cursor: pointer;
}
.authForm button:hover {
    background: #3bc9db;
}

.footer {
    margin-top: 2rem;
    text-align: right;
}
.footer a {
    text-decoration: underline;
    color: gray;
}
.footer a:hover {
    color: #bebebe;
}

.errorMessage {
    color: red;
    text-align: center;
    font-size: 0.875rem;
    margin-top: 1rem;
}
```

- **회원가입 프로그램**

[ex06]-[client]-[src]-[components]-[auth]-[Register.js]

```
import React, { useState } from 'react';
import {Link} from 'react-router-dom';
import '../../css/Auth.css';
import axios from 'axios';

const Auth = () => {
    const [error, setError] = useState('');
    const [form, setForm] = useState({
        userid: '',
        password: '',
        passwordConfirm: ''
    });

    const { userid, password, passwordConfirm } = form;

    const onChange = (e) => {
        const nextForm = {
            ...form,
            [e.target.name]: e.target.value
        };
        setForm(nextForm);
    };

    const onSubmit = (e) => {
        e.preventDefault();
        setError('');
        if(userid ==='' || password === '' || passwordConfirm === '') {
            setError('빈칸을 모두 입력하세요.');
            return;
        }
        if(password !== passwordConfirm){
            setError('비밀번호가 일치하지 않습니다.');
            return;
        }

        const url = '/auth/register';
        const data = {'userid': userid, 'password': password}
        axios.post(url, data)
            .then(response => {
                console.log('결과:' + response.data.result);
                if(response.data.result === 1) {
                    setError('이미 존재하는 계정입니다.');
                }else if(response.data.result === 0){
                    setError('성공!!!');
                }
        });
    }

    return (
        <div className="authBlock">
            <div className="whiteBox">
                <div className="logo-area">
                    <Link to='/'>REACTERS</Link>
                </div>
                <div className="authForm">
                    <h3>회원가입</h3>
                    <form onSubmit={onSubmit}>
                        <input type="text" placeholder="이이디" name="userid" value={userid} onChange={onChange}/>
                        <input type="password" placeholder="비밀번호" name="password" value={password}
                                                                                onChange={onChange}/>
                        <input type="password" placeholder="비밀번호확인" name="passwordConfirm" value={passwordConfirm}
                                                                                onChange={onChange}/>
                        <button type="submit">회원가입</button>
                    </form>
                    {error && <div className="errorMessage">{error}</div>}
                    <div className="footer"><Link to="/login">로그인</Link></div>
                </div>
            </div>
        </div>
    );
};

export default Auth;
```

- **머리글(Header) 프로그램**

[ex6]-[client]-[src]-[components]-[common]-[Header.js]

```
import React, { useState, useEffect } from 'react';
import '../../css/Header.css';
import { Link } from 'react-router-dom';

const Header = () => {
    const [userid, setUserid] = useState('');

    useEffect(() => {
       setUserid(sessionStorage.getItem('userid'));
    }, [userid]);

    const onLogout = () => {
        sessionStorage.removeItem('userid');
        setUserid('');
    }

    return (
        <div className="headerBlock">
            <div className="logo">
                <Link to ='/'>REACTERS</Link>
            </div>
            { userid ? (
                <div className="right">
                    <span className="userinfo">{userid}</span>
                    <button className="grayButton" onClick={onLogout}>로그아웃</button>
                </div>
            ) : (
                <div className="right">
                    <Link to = '/login'>
                        <button className="grayButton">로그인</button>
                    </Link>
                </div>
            ) }
        </div>
    );
};

export default Header;
```

[ex06]-[client]-[css]-[Header.css]

```
.headerBlock {
    overflow: hidden;
    width: 100%;
    background: white;
    box-shadow: 5px 5px 5px gray;
    padding: 1rem;
    margin-bottom: 10px;
}

.logo {
    float: left;
    font-size: 1.125rem;
    font-weight: 800;
    letter-spacing: 2px;
}

.right {
    margin-right:1rem;
    float: right;
}

.right .userinfo {
    margin-right:1rem;
    font-weight: 800;
}
```

- **게시글 목록 프로그램**

[ex06]-[client]-[src]-[pages]-[PostListPage.js]

```
import React from 'react';
import Header from '../components/common/Header';
import PostList from '../components/post/PostList';
const PostListPage = () => {
    return (
        <div>
            <Header/>
            <PostList/>
        </div>
    );
};
export default PostListPage;
```

[src]-[client]-[css]-[PostList.css]

```
.postListBlock {
    margin-top: 1rem;
    overflow: hidden;
}
.buttonWrapper {
    overflow: hidden;
    margin-bottom: 1rem;
    text-align: right;
}
.postItemBlock {
    padding: 2rem;
    border-bottom: 1px solid #e2e2e2;
}
.postItemBlock .subInfo {
    margin-top: 0.5rem;
    color: #8a8585;
}
.postItemBlock h2 {
    font-size: 2rem;
    margin-top: 1rem;
    margin-bottom: 0;
}
.postItemBlock h2 a {
    color: gray;
}
.postItemBlock h2 a:hover {
    color: #bebebe;
}
.postItemBlock p {
    margin-top: 2rem;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    -webkit-line-clamp: 2;
    -webkit-box-orient: vertical;
}
.subInfo span {
    padding-left: 0.25rem;
    padding-right: 0.25rem;
}
```

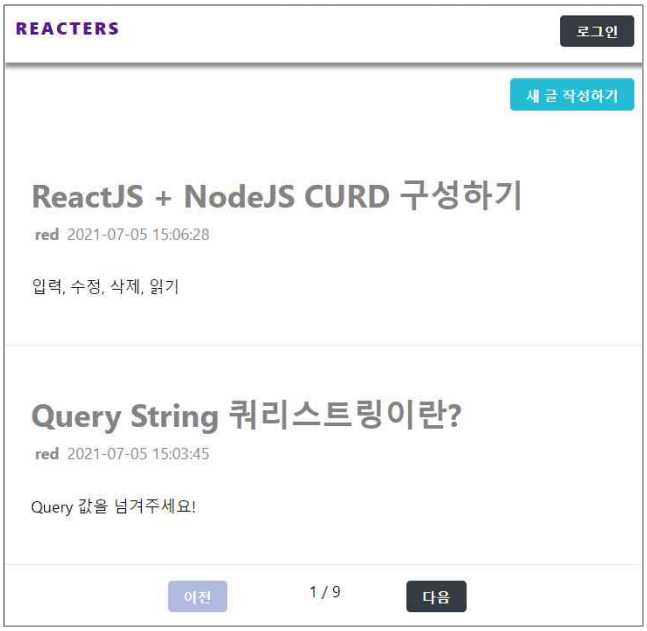
[ex06]-[client]-[css]-[Pagination.css]

```
.paginationBlock {
    width: 320px;
    margin: 0 auto;
    display: flex;
    justify-content: space-between;
    margin-bottom: 3rem;
    margin-top: 1rem;
}
.paginationBlock button:disabled {
    background: #adb5db;
}
```

쿼리 문자열을 객체로 변환을 위한 qs라는 라이브러리를 아래와 같이 설치한다.

```
yarn add qs
```

[ex06]-[client]-[src]-[components]-[PostList.js]




```
import React from 'react';
import '../../css/PostList.css';
import { useState, useEffect } from 'react';
import axios from 'axios';
import { Link, withRouter } from 'react-router-dom';
import Pagination from './Pagination';
import qs from 'qs';

const PostItem = ({id, title, body, fmtCreateDate, userid}) => {
    return (
        <div className="postItemBlock">
            <h2><Link to={`/view/${id}`}>{title}</Link></h2>
            <div className="subInfo">
                <span><b>{userid}</b></span>
                <span>{fmtCreateDate}</span>
            </div>
            <p>{body}</p>
        </div>
    );
};

const PostList = ({ location }) => {
    const [posts, setPosts] = useState(null);
    const [loading, setLoading] = useState('');
    const [userid, setUserid] = useState('');
    const [lastPage, setLastPage] = useState(0);
    const query = qs.parse(location.search, {ignoreQueryPrefix: true});
    const page = !query.page ? 1: query.page;

    useEffect(() => {
        const fetchData = async() => {
            setLoading(true);
            try {
                var url = `/post?page=${page}`;
                var response = await axios.get(url);
                setPosts(response.data);
                url = '/post/count';
                response = await axios.get(url);
                setLastPage(Math.ceil(parseInt(response.data.count)/2));
            }catch(e) {
                console.log('error...' + e);
            }
            setLoading(false);
        };
        fetchData();
        setUserid(sessionStorage.getItem('userid'));
    }, [page]);

    if( loading ) return <div>로딩중입니다...</div>;
    return (
        <div className="postListBlock">
            <div className="buttonWrapper">
                <Link to={userid ? '/write' : '/login'}><button className="skyButton">새 글 작성하기</button></Link>
            </div>
            <div>
                { posts && posts.map(p => (
                    <PostItem key={p.id} id={p.id} title={p.title} body={p.body}
                                                        fmtCreateDate={p.fmtCreateDate} userid={p.userid}/>
                )) }
            </div>
            <Pagination page={page} lastPage={lastPage}/>
        </div>
    );
};

export default withRouter(PostList);
```

[ex06]-[client]-[src]-[components]-[Pagination.js]

```
import '../../css/Pagination.css';
import { Link } from 'react-router-dom';
const Pagination = ({page, lastPage}) => {
    var intPage = parseInt(page);
    return (
        <div className="paginationBlock">
            <Link to={intPage === 1 ? '/?page=1' : `/?page=${intPage-1}`}>
                <button  className="grayButton" disabled={intPage === 1}>이전</button>
            </Link>
            <span>{page} / {lastPage}</span>
            <Link to={intPage === lastPage ? '?page=5': `/?page=${intPage+1}`}>
                <button className="grayButton" disabled={intPage === lastPage}>다음</button>
            </Link>
        </div>
    );
};
export default Pagination;
```

- **게시글 쓰기 프로그램**

[ex06]-[client]-[src]-[pages]-[WritePage.js]

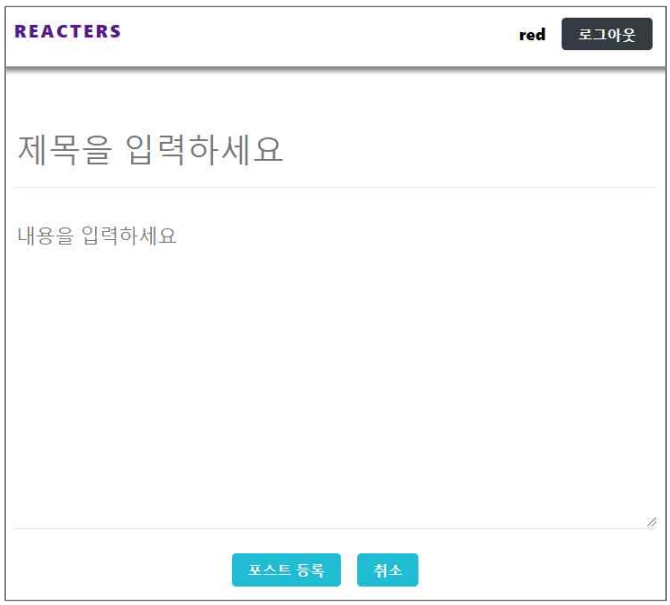
```
import React from 'react';
import Header from '../components/common/Header';
import PostWrite from '../components/post/PostWrite';
const WritePage = () => {
    return (
        <div>
           <Header/>
           <PostWrite/>
        </div>
    );
};
export default WritePage;
```

[ex06]-[client]-[css]-[PostWrite.css]

```
.postWriteBlock {
    overflow: hidden;
}
.postForm {
    padding: 1rem;
}
.postForm input {
    width: 100%;
    font-size: 2rem;
    border: none;
    outline: none;
    margin-top: 3rem;
    padding-bottom:1rem;
    border-bottom:1px solid #e2e2e2;
}
.postForm input:hover {
    border-bottom: 1px solid #495057;
}
.postForm textarea {
    font-family: inherit;
    width: 100%;
    font-size: 1.2rem;
    margin-top: 2rem;
    min-height: 300px;
    border: none;
    outline: none;
    border-bottom:1px solid #e2e2e2;
}
.postForm textarea:hover {
    border-bottom: 1px solid #495057;
}
.postButtonWrapper{
    overflow: hidden;
    text-align: center;
    padding-top: 20px;
}
```

[ex06]-[client]-[src]-[components]-[PostWrite.js]

```
import React from 'react';
import '../../css/PostWrite.css';
import { useState } from 'react';
import axios from 'axios';
import { withRouter } from 'react-router';

const PostWrite = ({ history }) => {
    const [form, setForm] = useState({
        title: '',
        body: ''
    });

    const { title, body } = form;

    const onChange = (e) => {
        const nextForm = {
            ...form,
            [e.target.name]: e.target.value
        };
        setForm(nextForm);
    }

    const onSubmit = (e) => {
        e.preventDefault();
        if(title === '') {
            alert('제목을 입력하세요!');
            return;
        }else if(body === ''){
            alert('내용을 입력하세요!');
        }else {
            if(!window.confirm('저장하실래요?')) return;

            const url = '/post/write';
            const data = {'userid': sessionStorage.getItem('userid'), 'title': title, 'body': body};
            axios.post(url, data).then(response => {
                if(response.data.result === 1) {
                    alert('새로운 글이 등록되었습니다.');
                    history.push('/');
                }
            });
        }
    }

    return (
        <div className="postWriteBlock">
            <form className="postForm" onSubmit={ onSubmit }>
                <input type="text"
                    placeholder="제목을 입력하세요!"
                    name="title"
                    value={title}
                    onChange={onChange}/>
                <textarea
                    placeholder="내용을 입력하세요!"
                    name="body"
                    onChange={onChange}>
                    { body }
                </textarea>
                <div className="postButtonWrapper">
                    <button className="skyButton" type="submit">포스트 등록</button>
                    <button className="skyButton" type="reset">취소</button>
                </div>
            </form>
        </div>
    );
};

export default withRouter(PostWrite);
```



- **게시글 읽기 프로그램**

[ex06]-[client]-[src]-[pages]-[ViewerPage.js]

```
import React from 'react';
import Header from '../components/common/Header';
import PostViewer from '../components/post/PostViewer';

const ViewerPage = () => {
    return (
        <div>
            <Header/>
            <PostViewer/>
        </div>
    );
};

export default ViwerPage;
```

[exx06]-[client]-[css]-[PostViewer.css]

```
.viewerWriteBlock {
    overflow: hidden;
}

.viewerForm {
    padding: 1rem;
}
.viewerForm input {
    width: 100%;
    font-size: 2rem;
    border: none;
    outline: none;
    margin-top: 3rem;
    padding-bottom:1rem;
}

.viewerForm input:hover {
    border-bottom: 1px solid #495057;
}

.viewerForm textarea {
    font-family: inherit;
    width: 100%;
    font-size: 1.2rem;
    margin-top: 2rem;
    min-height: 300px;
    border: none;
    outline: none;
    border-bottom:1px solid #e2e2e2;
}

.viewerForm textarea:hover {
    border-bottom: 1px solid #495057;
}

.viewerForm .subInfo {
    margin-top: 1rem;
    color: #495057;
    padding-bottom: 1rem;
    border-bottom: 1px solid #e2e2e2;
}
```

[ex06]-[client]-[src]-[components]-[PostViewer.js]

```
import React, { useState, useEffect } from 'react';
import '../../css/PostViewer.css';
import axios from 'axios';
import { withRouter } from 'react-router';
import PostUpdate from './PostUpdate';

const PostViewer = ({match}) => {
    const { postid } = match.params;
    const loginid = sessionStorage.getItem('userid');

    const [form, setForm] = useState({
        id: 0,
        title: '',
        userid: '',
        body: '',
        fmtCreateDate: ''
    });
    const {id, title, userid, body, fmtCreateDate} = form;
    const onChange = (e) => {
        const nextForm = {
            ...form,
            [e.target.name]: e.target.value
        };
        setForm(nextForm);
    }

    useEffect(() => {
        const fetchData = async() => {
            try{
                const url=`/post/view/${postid}`;
                const response =await axios.get(url);
                setForm(response.data);
            }catch(e){
                console.log('error...' + e);
            }
        }
       fetchData();
    }, [postid]);

    return (
        <div className="postViewerBlock">
            <form className="viewerForm">
                <input
                    type="text"
                    placeholder="제목을 입력하세요!"
                    name="title"
                    value={title}
                    onChange={onChange}
                    disabled={userid!==loginid && true}/>
                <div className="subInfo">
                    <span><b>{userid}</b></span>
                    <span>{fmtCreateDate}</span>
                </div>
                { userid === loginid && <PostUpdate id={id} title={title} body={body}/> }
                <textarea
                    placeholder="내용을 입력하세요!"
                    name="body"
                    value={body}
                    onChange={onChange}
                    disabled={userid!==loginid && true}>
                </textarea>
            </form>
        </div>
    );
};

export default withRouter(PostViewer);
```

- **게시글 수정, 삭제 프로그램**

[ex06]-[client]-[css]-[PostUpdate.css]

```
.postUpdate {
    overflow: hidden;
    margin-top: 1rem;
    text-align: right;
}

.postUpdate span {
    font-size: 0.875rem;
    color: #495057;
    padding: 0.3rem;
    cursor: pointer;
}

.postUpdate span:hover {
    color: #e2e2e2;
}
```

[ex06]-[client]-[src]-[components]-[PostUpdate.js]

```
import React from 'react';
import '../../css/PostUpdate.css';
import axios from 'axios';
import { withRouter } from 'react-router-dom';

const PostUpdate = ({ history, id, title, body }) => {
    const onDelete = () => {
        if(!window.confirm('삭제하실래요?')) return;

        const url = '/post/delete';
        const data = { 'id': id };

        axios.post(url, data).then(response => {
            if(response.data.result === 1) {
                alert('글이 삭제되었습니다.');
                history.push('/');
            }
        });
    }

    const onUpdate = () => {
        if(title === '') {
            alert('제목을 입력하세요!');
            return;
        }else if(body === ''){
            alert('내용을 입력하세요!');
        }else {
            if(!window.confirm('수정하실래요?')) return;
            const url = '/post/update';
            const data = { 'id': id, 'title': title, 'body': body };

            axios.post(url, data).then(response => {
                if(response.data.result === 1) {
                    alert('글이 수정되었습니다.');
                    history.push('/');
                }
            });
        }
    }

    return (
        <div className="postUpdate">
            <span onClick={ onUpdate }>수정</span>
            <span onClick={ onDelete }>삭제</span>
        </div>
    );
};

export default withRouter(PostUpdate);
```

# 15장 리덕스를 이용한 블로그 페이지 프로그래밍

- **리덕스 관련 라이브러리 설치**

```
yarn add redux react-redux  리덕스와 react-redux 라이브러리를 설치
```

```
yarn add redux-logger  브라우저 콘솔에 깔끔한 형식으로 출력하기 위해 redux-logger를 설치
```

```
yarn add redux-thunk  thunk는 특정 작업을 나중에 할 수 있도록 미루기 위해 함수 형태로 감싸준다.
```

- **액션함수 리듀서 작성 및 적용**

[ex06]-[client]-[src]-[index.js]

```
...
import { Provider } from 'react-redux';
import { applyMiddleware, createStore } from 'redux';
import rootReducer from './modules/rootReducer';
import { createLogger } from 'redux-logger';
import ReduxTunk from 'redux-thunk';

const logger = createLogger();
const store = createStore(rootReducer, applyMiddleware(logger, ReduxTunk));

ReactDOM.render(
    <Provider store ={ store }>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </Provider>,
    document.getElementById('root')
);
```

[ex06]-[client]-[src]-[App.js]

```
...
/* 리덕스 페이지 */
import ReduxWrite from './pages/redux/ReduxWrite';
import ReduxList from './pages/redux/ReduxList';
import ReduxViewer from './pages/redux/ReduxViewer';

function App() {
    return (
        <div className="App">
            ...
            { /* 리덕스 페이지 */ }
            <Route component={ReduxWrite} path='/rwrite'/>
            <Route component={ReduxList} path='/rList'/>
            <Route component={ReduxViewer} path='/rView/:postid'/>
        </div>
    );
}
export default App;
```

[ex06]-[client]-[src]-[modules]-[rootReducer.js]

```
import { combineReducers } from 'redux';
import write from './write';
import list from './list';
import viewReducer from './view';
import updateReducer from './update';

const rootReducer = combineReducers({
    write,
    list, view:
    viewReducer,
    update: updateReducer
});
export default rootReducer;
```

[ex06]-[client]-[src]-[modules]-[list.js]

```
import axios from "axios";

//액션 타입 정의
const LIST_POST_REQUEST = 'LIST_POST_REQUEST';
const LIST_POST_SUCCESS = 'LIST_POST_SUCCESS';
const LIST_POST_FAILURE = 'LIST_POST_FAILURE';
const COUNT_POST_SUCCESS = 'COUNT_POST_SUCCESS';

//액션 생성 함수
const listRequest = () => {
    return { type: LIST_POST_REQUEST, }
}
const listSuccess = (posts, lastPage) => {
    return {
        type: LIST_POST_SUCCESS,
        payload: { posts: posts, lastPage: lastPage }
    }
}

const listFaiure = (error) => {
    return { type: LIST_POST_FAILURE, payload: error }
}

export const listPost = (page) => {
    return async(dispatch) => {
        dispatch(listRequest());
        try {
            var url = `/post?page=${page}`;
            var response = await axios.get(url);
            const posts = response.data;
            url = '/post/count';
            response = await axios.get(url);
            const lastPage = Math.ceil(parseInt(response.data.count)/2);
            dispatch(listSuccess(posts, lastPage));
        }catch(e) {
            dispatch(listFaiure(e));
        }
    }
}

const initialState = {
    posts: [],
    loading: false,
    error: null,
    lastPage: 1,
}

const list = (state=initialState, action) => {
    switch(action.type){
        case LIST_POST_REQUEST:
            return {
                ...state, loading: true
            }
        case LIST_POST_SUCCESS:
            return {
                ...state, loading: false, posts: action.payload.posts, lastPage: action.payload.lastPage
            }
        case COUNT_POST_SUCCESS:
            return {
                ...state, count: action.payload
            }
        case LIST_POST_FAILURE:
            return {
                ...state, loading: true, error: action.payload
            }
        default:
            return state;
    }
}
export default list;
```

[ex06]-[client]-[src]-[modules]-[write.js]

```
import axios from "axios";
//액션 타입 정의
const WRITE_POST_REQUEST = 'WRITE_POST_REQUEST';
const WRITE_POST_SUCCESS = 'WRITE_POSST_SUCCESS';
const WRITE_POST_FAILURE = 'WRITE_POST_FAILURE';

const POST_CHANGE_FIELD = 'POST_CHANGE_FIELD';
const POST_INITIAL = 'POST_INITIAL';

//액션 생성 함수
export const changeField = (e) => {
    return {
        type: POST_CHANGE_FIELD,
        payload: { name: e.target.name, value: e.target.value }
    }
}

export const postInitial = () => {
    return { type: POST_INITIAL }
}

export const postInsert = (userid, title, body) => {
    return (dispatch) => {
        dispatch(postRequest);
        const url = '/post/write';
        const data = {'userid': userid, 'title': title, 'body': body};
        axios.post(url, data)
            .then(response => dispatch(postSuccess(response.data.result)))
            .catch(error => dispatch(postFaiure(error)));
    }
}

const postRequest = () => {
    return { type: WRITE_POST_REQUEST }
}
const postSuccess = (result) => {
    return { type: WRITE_POST_SUCCESS, payload: result }
}

const postFaiure = (error) => {
    return { type: WRITE_POST_FAILURE, payload: error }
}

//리듀서 작성하기
const initialState = { title: '', body: '', result: null, loading: false, error: null };

const write = (state=initialState, action) => {
    switch(action.type) {
        case POST_INITIAL:
            return {
                ...initialState
            }
        case POST_CHANGE_FIELD:
            return {
                ...state,
                [action.payload.name]: action.payload.value
            }
        case WRITE_POST_REQUEST:
            return {
                ...state,
                loading: true
            }
        case WRITE_POST_SUCCESS:
            return {
                ...state,
                loading: false,
                result: action.payload
            }
        case WRITE_POST_FAILURE:
            return {
                ...state,
                error: action.payload,
                loading: false
            }
        default: return state;
    }
}
export default write;
```

[ex06]-[client]-[src]-[modules]-[view.js]

```
import axios from 'axios';

//액션 타입 정의
const VIEW_POST_REQUEST = 'VIEW_POST_REQUEST';
const VIEW_POST_SUCCESS = 'VIEW_POSST_SUCCESS';
const VIEW_POST_FAILURE = 'VIEW_POST_FAILURE';
const POST_CHANGE_FIELD = 'POST_CHANGE_FIELD';

//액션 생성 함수
export const changeField = (e) => {
    return {
        type: POST_CHANGE_FIELD,
        payload: { name: e.target.name, value: e.target.value }
    }
}
const viewRequest = () => {
    return { type: VIEW_POST_REQUEST }
}
const viewSuccess = (result) => {
    return {
        type: VIEW_POST_SUCCESS,
        payload: { post: result, title: result.title, body: result.body }
    }
}
const viewPostFaiure = (error) => {
    return {
        type: VIEW_POST_FAILURE,
        payload: error
    }
}
export const viewPost = (id) => {
    return async(dispatch) => {
        dispatch(viewRequest());
        try {
            var url = `/post/view/${id}`;
            var response = await axios.get(url);
            dispatch(viewSuccess(response.data))
        }catch(e) {
            dispatch(viewPostFaiure(e));
        }
    }
}

//리듀서 작성하기
const initialState = { post: null, title: '', body: '', loading: false, error: null, success: false };

const viewReducer = (state=initialState, action) => {
    switch(action.type){
        case POST_CHANGE_FIELD:
            return {
                ...state,
                [action.payload.name]: action.payload.value
            }
        case VIEW_POST_REQUEST:
            return {
                ...state,
                loading: true
            }
        case VIEW_POST_SUCCESS:
            return {
                ...state,
                loading: false,
                post: action.payload.post,
                title: action.payload.title,
                body: action.payload.body
            }
        case VIEW_POST_FAILURE:
            return {
                ...state,
                loading: false,
                error: action.payload
            }
        default:
        return state;
    }
}

export default viewReducer;
```

[ex06]-[client]-[src]-[modules]-[update.js]

```
import axios from "axios";

//액션 타입 정의
const POST_REQUEST = 'UPDATE_POST_REQUEST';
const POST_SUCCESS = 'UPDATE_POSST_SUCCESS';
const POST_FAILURE = 'UPDATE_POST_FAILURE';

//액션 생성 함수
export const postDelete = (id) => {
    return (dispatch) => {
        dispatch(postRequest);
        const url = '/post/delete';
        const data = { 'id': id };
        axios.post(url, data)
            .then(response => dispatch(postSuccess(response.data.result)))
            .catch(error => dispatch(postFaiure(error)));
    }
}

export const postUpdate = (id, title, body) => {
    return (dispatch) => {
        dispatch(postRequest);
        const url = '/post/update';
        const data = { 'id': id, 'title': title, 'body': body };
        axios.post(url, data)
            .then(response => dispatch(postSuccess(response.data.result)))
            .catch(error => dispatch(postFaiure(error)));
    }
}

const postRequest = () => {
    return {
        type: POST_REQUEST
    }
}

const postSuccess = (result) => {
    return {
        type: POST_SUCCESS,
        payload: result
    }
}

const postFaiure = (error) => {
    return {
        type: POST_FAILURE,
        payload: error
    }
}

//리듀서 작성하기
const initialState = { title: '', body: '', result: null, loading: false, error: null, };

const updateReducer = (state=initialState, action) => {
    switch(action.type) {
        case POST_REQUEST:
            return {
                ...state,
                loading: true
            }
        case POST_SUCCESS:
            return {
                ...state,
                loading: false,
                result: action.payload
            }
        case POST_FAILURE:
            return {
                ...state,
                error: action.payload,
                loading: false
            }
        default:
            return state;
    }
}

export default updateReducer;
```

- **블로그 목록 프로그램**

[ex06]-[client]-[src]-[pages]-[redux]-[ReduxList.js]

```
import React, { useEffect } from 'react';
import { connect } from 'react-redux';
import { listPost } from '../../modules/list';
import RPagenation from './RPagenation';
import { withRouter } from 'react-router-dom';
import qs from 'qs';
import { Link } from 'react-router-dom';

const PostItem = ({ id, title, body, page }) => {
    return (
        <div>
            <h2><Link to={ `/rview/${id}?page=${ page }` }>{ title }</Link></h2>
            <p>{body}</p>
            <hr/>
        </div>
    )
};

const ReduxList = ({ location, listPost, loading, posts, lastPage }) => {
    const query = qs.parse(location.search, { ignoreQueryPrefix : true });
    const curPage = !query.page ? 1: query.page;
    useEffect(() => {
        listPost(curPage);
     }, [curPage]);
    return (
        <div>
            <h1>[글 목록]</h1>
            <Link to='/rwrite'>글쓰기</Link>
            <hr/>
            { loading ? (<div>로딩중</div>): (
                posts.map(post=>(
                    <PostItem key={ post.id } id={ post.id } title={ post.title } body={ post.body } page={ curPage }/>
                ))
            )}
            <RPagenation curPage={ curPage } lastPage={ lastPage }/>
        </div>
    );
};

const mapStateToProps = ({ list }) => {
    return {
        posts: list.posts,
        loading: list.loading,
        lastPage: list.lastPage
    }
}
const mapDispatchToProps = {
    listPost,
}
export default withRouter(connect(mapStateToProps, mapDispatchToProps)(ReduxList));
```

[ex06]-[client]-[src]-[pages]-[redux]-[RPagenation.js]

```
import React from 'react';
import { Link } from 'react-router-dom';

const RPagenation = ({ curPage, lastPage }) => {
    var page = parseInt(curPage);
    return (
        <div>
            <Link to={ `/rlist?page=${ page-1 }` }><button disabled={ page === 1 }>이전</button></Link>
            <span> { page }/{ lastPage }  </span>
            <Link to={ `/rlist?page=${ page+1 }` }><button disabled={ page === lastPage }>다음</button></Link>
        </div>
    );
};

export default RPagenation;
```

- **블로그 쓰기 프로그램**

[ex06]-[client]-[src]-[pages]-[redux]-[ReduxWrite.js]

```
import React, { useEffect } from 'react';
import { changeField, postInsert, postInitial } from '../../modules/write';
import { connect } from 'react-redux';
import { withRouter } from 'react-router';

const ReduxWrite = ( { history, title, body, changeField, postInsert, result, postInitial }) => {
    const onChange = (e) => {
        changeField(e);
    }
    const onSubmit = async(e) => {
        e.preventDefault();
        if(!window.confirm(title + '등록하실래요?')) return;
        postInsert('red', title, body);
    }
    useEffect(() => {
        alert(result);
        if(result) {
            alert('새로운 글이 등록되었습니다!');
            postInitial();
            history.push('/rlist');
        }
    }, [result]);

    return (
        <div>
            <h1>[글쓰기]</h1>
            <form onSubmit={ onSubmit }>
                제목: <input type="text" name="title" value={ title } onChange={ onChange } size="50"/><hr/>
                내용: <textarea name="body" value={ body } onChange={ onChange } cols="52" rows="10"></textarea><hr/>
                <button type="submit">글쓰기</button>
            </form>
        </div>
    );
};

const mapStateToProps = ({ write }) => {
    return { title: write.title, body: write.body, result: write.result }
}
const mapDispatchToProps = {
    changeField: (e) => changeField(e),
    postInsert: (userid, title, body) => postInsert(userid, title, body),
    postInitial
}
export default withRouter(connect(mapStateToProps, mapDispatchToProps)(ReduxWrite));
```

- **블로그 읽기 프로그램**

[ex06]-[client]-[src]-[pages]-[redux]-[ReduxViewer.js]

```
import React, { useEffect } from 'react';
import { viewPost, changeField } from '../../modules/view';
import { connect } from 'react-redux';
import { withRouter } from 'react-router';
import ReduxUpdate from './ReduxUpdate';
import qs from 'qs';

const ReduxViewer = ({ match,  location, viewPost, post, loading, changeField, title, body }) => {
    const { postid } = match.params;
    const query = qs.parse(location.search, { ignoreQueryPrefix: true });
    const curPage = !query.page ? 1: query.page;

    useEffect(() => {
        viewPost(postid);
    },[]);

    const onChange = (e) => {
        changeField(e);
    }
```

```
    return (
        <div>
            { loading || !post ? (<div>로딩중...</div>) : (
                <div>
                    <h1>[글읽기]</h1>
                    <form>
                        제목: <input type="text" name="title" size="50" value={ title } onChange={ onChange }/><hr/>
                        작성일: { post.fmtCreateDate }<hr/>
                        내용: <textarea name="body" cols="52" rows="10" value={ body } onChange={ onChange }></textarea>
                    </form><hr/>
                    <ReduxUpdate id={ postid } title={ title } body={ body } page={ curPage }/>
                </div>
            )}
        </div>
    );
};

const mapStateToProps = ({ view }) => {
    return { post: view.post, loading: view.loading, title: view.title, body: view.body }
}
const mapDsipatchToPorps ={
    viewPost,
    changeField: (e) => changeField(e),
}
export default withRouter(connect(mapStateToProps, mapDsipatchToPorps)(ReduxViewer));
```

- **블로그 수정 삭제 프로그램**

[ex06]-[client]-[src]-[pages]-[redux]-[ReduxUpdate.js]

```
import React from 'react';
import { connect } from 'react-redux';
import { postDelete, postUpdate } from '../../modules/update';
import { withRouter } from 'react-router-dom';

const ReduxUpdate = ({ id, title, body, postDelete, postUpdate, history, page }) => {
    const onClickUpdate = () => {
        if(title === '') {
            alert('제목을 입력하세요!');
            return;
        }else if(body === ''){
            alert('내용을 입력하세요!');
        }else {
            if(!window.confirm('수정하실래요?')) return;
            postUpdate(id, title, body);
            alert('글이 수정되었습니다.');
            history.push(`/rlist?page=${page}`);
        }
    }
    const onClickDelete = () => {
        if(!window.confirm(id + '삭제하실래요?')) return;
        postDelete(id);
        alert('글이 삭제되었습니다.');
        history.push('/rlist');
    }

    return (
        <div>
            <button onClick={ onClickUpdate }>수정</button><button onClick={ onClickDelete }>삭제</button>
        </div>
    );
};

const mapStateToProps = ({update}) => {
    return { loading: update.loading, }
}
const mapDsipatchToPorps = {
    postDelete: (id) => postDelete(id),
    postUpdate: (id, title, body) => postUpdate(id, title, body)
}
export default withRouter(connect(mapStateToProps, mapDsipatchToPorps)(ReduxUpdate));
```

# 16장 기타

- **무한 스크롤링**

[src]-[components]-[Photos.js]

```
import {useEffect, useState, useRef} from 'react';

const Photos = () => {
    const [loading, setLoading] = useState(false);
    const [data, setData] = useState([]);
    const [page, setPage] = useState(1);
    const lastRef = useRef(null);
    const [lastPage, setLastPage] = useState(10);

    const callAPI = () => {
        setLoading(true);
        fetch('https://jsonplaceholder.typicode.com/photos')
        .then(response => response.json())
        .then(json => {
            const newData = json.filter(j => j.id >= (page-1)*10+1 && j.id <= (page*10));
            setData(data.concat(newData));
            setPage(page + 1);
            setLoading(false);
        });
    }
    //onIntersect에서는 현재 관찰하고 있는 부분을 해제하고 callAPI()를 호출한 후 새롭게 밑부분이 된 부분을 관찰한다.
    const onIntersect = ([entry], observer) => {
        if (entry.isIntersecting) {
            observer.unobserve(entry.target);
            callAPI();
            observer.observe(lastRef.current);
        }
    };
    //첫 화면의 경우 loading이 false값이고 빈 공간이다. 또한 페이지 끝이므로 Intersecting이 관찰되어 onIntersect를 호출하게 된다.
    useEffect(() => {
        if(!loading && page <= lastPage) {
            const observer = new IntersectionObserver(onIntersect);
            observer.observe(lastRef.current);
            return () => observer.disconnect();
        }
      }, [loading]);

    return (
        <div>
            {data ? data.map(d=>(
                <div style={{marginBottom:'10px'}}>
                    <h3>{d.id}:{d.title}</h3>
                </div>
            )): ''}
            <div ref={lastRef}><b>{loading && 'Loading...'}</b></div>
        </div>
    );
};

export default Photos;
```

[src]-[App.js]

```
import Photos from './Photos';
import './App.css';
function App() {
    return(
        <div className="App">
            <div className="box">
                <Photos/>
            </div>
      </div>
    );
}
export default App;
```

```
.App {
    background: aqua;
    padding: 50px;
}
.box {
    width: 800px;
    background: white;
    padding: 20px;
    height: 500px;
    overflow: auto;
}
```