

# React 애플리케이션 개발

---

ES6로 개발하는 ReactJS

장민창

mcjang@huccloud.co.kr

# React 개요

---

1. 리액트 프레임워크
2. 개발환경 세팅
3. 리액트 애플리케이션 생성

# 1. 리액트 프레임워크

# 1. 리액트 프레임워크

---

- 리액트 (React)
  - Open-source javascript framework
  - 2013년 페이스북(현재 메타)에서 개발/발표
- SPA 개발 프레임워크
  - SPA (Single Page Application) 기반의 프레임워크
- Javascript 기반의 프레임워크
  - ECMA Script 2015 (ES6) 이상의 문법만을 지원함.

The React logo, featuring the word "React" in a bold, blue, sans-serif font.

A JavaScript library for building user interfaces

# 1. 리액트 프레임워크

- SPA vs. MPA

**SPA**



**MPA**



# 1. 리액트 프레임워크

---

- MPA (Multi Page Application)
  - Web Application (WebApp) 이 여러 개의 페이지로 구성된 것
  - 화면의 요청이 있을 때 마다(URL) 새로운 페이지를 응답
    - 페이지가 새롭게 응답될 때 마다 깜박이는 현상 발생.
- SPA (Multi Page Application)
  - WebApp이 하나의 페이지로만 구성된 것.
  - 화면의 요청이 있을 때 마다(URL) 바뀌는 컴포넌트만 갱신
    - 페이지가 새롭게 응답될 때 깜박이는 현상이 없음

## 2. 개발환경 세팅

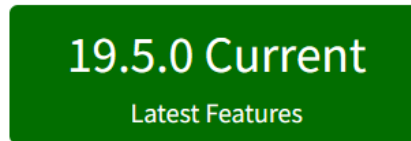
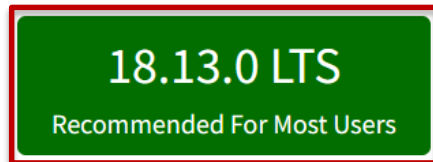
## 2. 개발환경 세팅

---

- node.js 설치 (<https://nodejs.org/>)

Node.js® is an open-source, cross-platform JavaScript runtime environment.

Download for Windows (x64)



[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

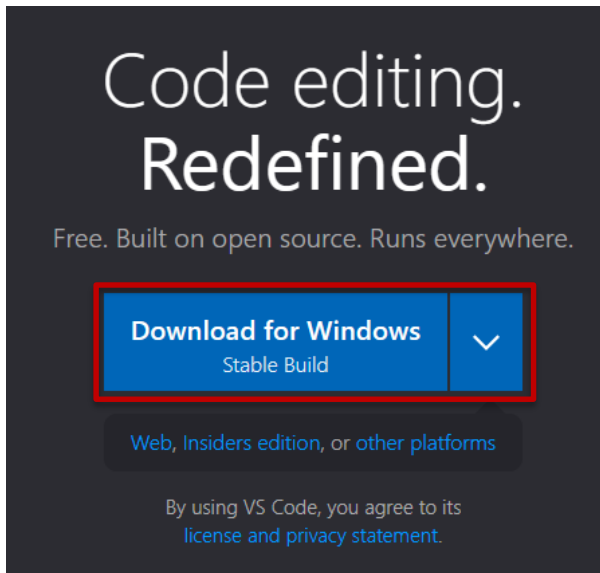
- Javascript 기반의 서버 프로그램
- nodejs 기반의 npm Repository를 활용하기 위해 설치 (React는 npm에 등록된 라이브러리)



## 2. 개발환경 세팅

---

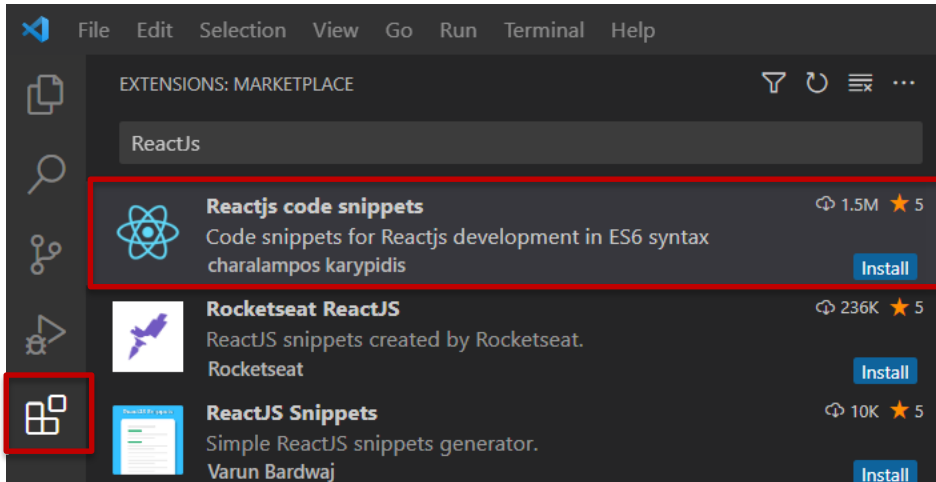
- VSCode 설치 (<https://code.visualstudio.com/>)



- 여러 언어들을 지원하는 경량 개발 툴
- 플러그인을 통해 다양하게 사용이 가능하다.

## 2. 개발환경 세팅

- Reactjs code snippets 플러그인 설치



- Reactjs Content Assist를 지원.

## 2. 개발환경 세팅

- 4. 크롬 브라우저 React Developer Tools 플러그인 설치

홈 > 확장 프로그램 > React Developer Tools



React Developer Tools

추천

★★★★★ 1,407 ⓘ | 개발자 도구 | 사용자 3,000,000+명

Chrome에 추가

- Reactjs debug 지원

### **3. 리액트 애플리케이션 생성**

### 3. 리액트 애플리케이션 생성

- Reactjs Project 생성
  - Command 창에서 아래와 같이 입력

```
> cd c:\W  
> mkdir reactProject  
> cd reactProject  
> npx create-react-app react-ui
```



```
C:\#reactProject>npx create-react-app react-ui  
Need to install the following packages:  
  create-react-app@5.0.1  
Ok to proceed? (y) _
```

- npx create-react-app 프로젝트명

# 3. 리액트 애플리케이션 생성

- Reactjs Project 실행
  - Command 창에서 아래와 같이 입력

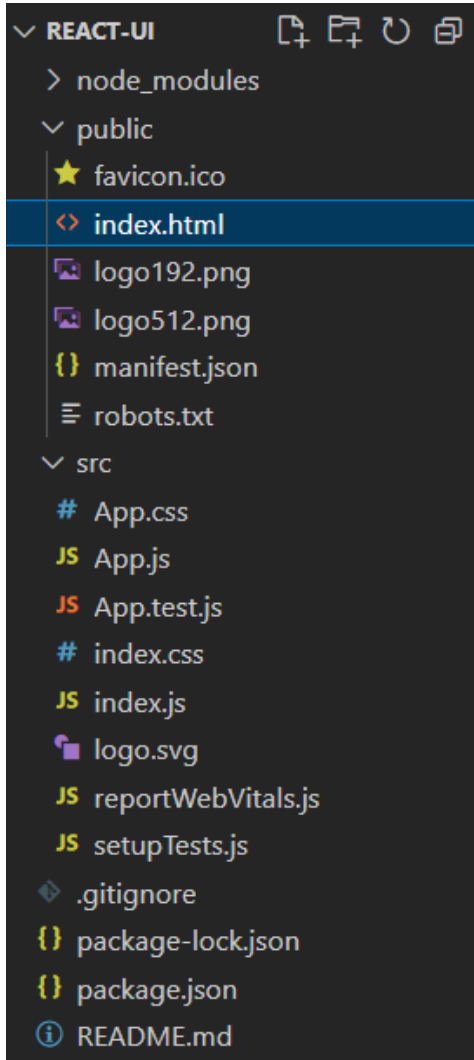
```
> cd react-ui  
> npm start
```

```
Compiled successfully!  
  
You can now view react-ui in the browser.  
  
Local:            http://localhost:3000  
On Your Network:  http://192.168.0.10:3000  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.  
  
webpack compiled successfully
```



# 3. 리액트 애플리케이션 생성

- React Application 구조



- node\_modules**
  - React App 개발에 필요한 외부 라이브러리들의 모음
  - package.json 에 의해 관리된다.
- index.html**
  - 메인 홈페이지 파일. 편집할 필요 x
- App.js**
  - React Root 컴포넌트 파일
- index.js**
  - React Starting Point 파일.
  - App.js 파일을 로딩해 웹 브라우저에 실행함.
- package.json**
  - node.js의 외부라이브러리 설정 파일

리액트 UI 구성

# Components

---

- 4. 컴포넌트 개요
- 5. 함수형 컴포넌트
- 6. 클래스 컴포넌트
- 7. 함수형 / 클래스
- 8. JSX 훑어보기



## 4. 컴포넌트 개요

## 4. 컴포넌트 개요

---

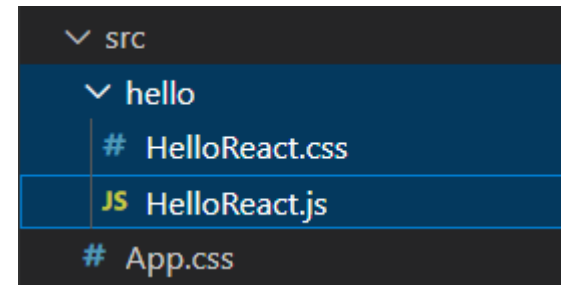
- 컴포넌트
  - 화면을 UI를 구성하는 클래스 또는 함수
- 리액트 컴포넌트
  - JSX문법으로 생성된 HTML Element(Tag) 에 대응하는 모듈
  - 필요에 따라 “함수형 컴포넌트”, “클래스 컴포넌트” 로 사용
- 리액트 컴포넌트 구분
  - 리액트에서 HTML Element는 첫 글자를 소문자로 작성
  - 리액트 컴포넌트는 첫 글자를 대문자로 작성
  - **컴포넌트의 첫 글자로 HTML Element와 컴포넌트를 구분한다.**

## 5. 함수형 컴포넌트

## 5. 함수형 컴포넌트

- 함수형 컴포넌트 생성
- ./hello/HelloReact.js

```
1  import './HelloReact.css';
2
3  export default function HelloReact() {
4    return (
5      <div id="hello">
6        <p>HelloReact!</p>
7      </div>
8    );
9  }
```



- ./hello/HelloReact.css

```
1  #hello {
2    font-size: 15px;
3    font-weight: bold;
4  }
```

## 5. 함수형 컴포넌트

- 함수형 컴포넌트 생성
- ./App.js

```
1  import HelloReact from './hello/HelloReact';
2
3  ∨ function App() {
4  ∨    return (
5      |    <HelloReact />
6      |    );
7  }
8
9  export default App;
10
```

← → ↻ ⓘ localhost:3000

**HelloReact!**

# 5. 함수형 컴포넌트

---

- import / export
  - 외부에서 컴포넌트를 사용(import) 할 수 있도록 export 필요.
  - export default
    - import MODULE from PATH;
  - export
    - import {MODULE1, MODULE2} from PATH;
- return ( Elements | Component );
  - 하나의 컴포넌트는 하나의 Root Element를 가져야만 한다.
    - 두 개 이상의 Root Element는 가질 수 없음.
  - 적당한 Element가 없을 경우
    - <> 내용 </>
    - <Fragment> 내용 </Fragment>

## 6. 클래스 컴포넌트


## 6. 클래스 컴포넌트

- 클래스 컴포넌트 생성
- ./hi/HiReact.js

```
1  import "../HiReact.css";
2  import { Component } from "react";
3
4  export class HiReact extends Component {
5
6      render() {
7          return (
8              <div id="hi-react">
9                  <p>HiReact!</p>
10             </div>
11         );
12     };
13
14 }
```

```
▼ src
  > hello
  ▼ hi
    # HiReact.css
    JS HiReact.js
```

- ./hi/HiReact.js

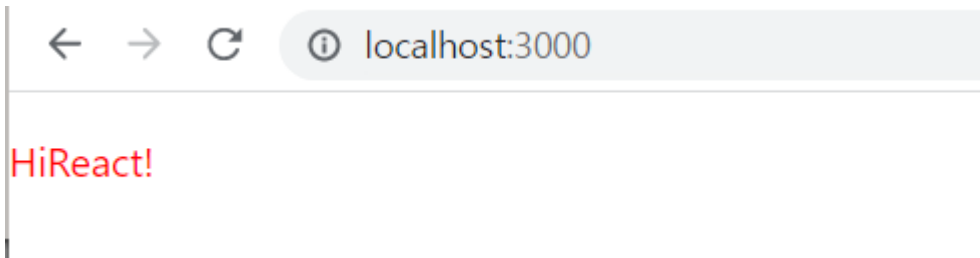
```
1  #hi-react {
2      color:  #FF0000;
3  }
```



## 6. 클래스 컴포넌트

- 클래스 컴포넌트 생성
- ./App.js

```
1  import { HiReact } from './hi/HiReact';
2
3  function App() {
4    return (
5      <HiReact />
6    );
7  }
8
9  export default App;
```



## 6. 클래스 컴포넌트

---

- 클래스 컴포넌트는 Component를 상속받아야 한다.
  - extends Component
- 클래스 컴포넌트는 render() 를 재정의 해야 함.
  - render 내에서 return ( 내용 ); 으로 컴포넌트를 구성함.

## 7. 함수형 / 클래스

## 7. 함수형 / 클래스

---

- 일반적인 컴포넌트는 Class 형태로 선언.
- 다음과 같은 조건이 있을 경우, 함수형 컴포넌트 사용 가능
  - 1. Lifecycle API를 사용하지 않을 경우
  - 2. State를 사용하지 않을 경우
  - 3. Props만 사용하는 경우
- React Hook
  - 함수형 컴포넌트에서 Lifecycle, State, Props 사용이 가능함.

## 8. JSX 훑어보기

## 8. JSX 훑어보기

---

- JSX = Javascript XML
  - Javascript에 XML을 추가한 확장형 언어
- Javascript를 이용해 UI를 구성한다.
  - HTML 문법과 유사해 빠르게 UI를 구성할 수 있음.

```
function App() {  
  return (  
    <div>  
      <HelloReact />  
      <HiReact />  
      <HiReact />  
      <HiReact />  
      <HiReact />  
    </div>  
  );  
}
```

## 8. JSX 훑어보기

- JSX의 모든 태그는 종료 태그가 필수.

```
function App() {  
  return (  
    <div>  
      <HelloReact></HelloReact>  
      <HiReact />  
    </div>  
  );  
}
```

- JSX는 단 하나의 Element만 리턴해야 함.

```
function App() {  
  return (  
    <HelloReact></HelloReact>  
    <HiReact />  
  );  
}
```

- 적당한 엘리먼트가 없을 경우 <> </> 혹은 <Fragment> </Fragment>

## 8. JSX 훑어보기

- JSX에서 변수 참조 및 JS 문법은 { }을 사용

```
export default function HelloReact() {  
  let message = "Hello!? React!?";  
  return (  
    <div id="hello">  
      <p>{message}</p>  
    </div>  
  );  
}
```

```
export class HiReact extends Component {  
  render() {  
    const message = "HiReact!?";  
  
    return (  
      <div id="hi-react">  
        <p>{message}</p>  
      </div>  
    );  
  };  
}
```

```
export class HiReact extends Component {  
  render() {  
    let list = ["Hello?", "React?"];  
  
    return (  
      <div id="hi-react">  
        {  
          list.map((row, idx) => <p key={idx} className={"para_" + idx}>{row}</p>)  
        }  
      </div>  
    );  
  };  
}
```



## 8. JSX 훑어보기

- DOM Event Handler는 Camelcase로만 작성한다.

```
export class HiReact extends Component {  
  render() {  
    let list = ["Hello?", "React?"];  
  
    return (  
      <div id="hi-react">  
        {  
          list.map((row, idx) => <p key={idx} onClick={(e) => console.log({row})}>{row}</p>)  
        }  
      </div>  
    );  
  };  
};  
}
```

## 8. JSX 훑어보기

- Element에 class 할당할 때엔 className 속성을 이용
  - class 가 예약어로 사용되기 때문
- 주석은 `{/* 주석내용 */}` 으로 작성
- Inline Style을 객체 형식으로 사용한다.

```
export class HiReact extends Component {
  render() {
    let list = ["Hello?", "React?"];

    return (
      <div id="hi-react" style={{ cursor: "pointer" }}>
        {
          list.map((row, idx) => <p key={idx} className={"para_" + idx}>{row}</p>)
        }
      </div>
    );
  }
}
```

부모 컴포넌트에서 자식 컴포넌트로 읽기전용 데이터 전달

# Property

---

9. 프로퍼티 (Props)

10. Component Body Props

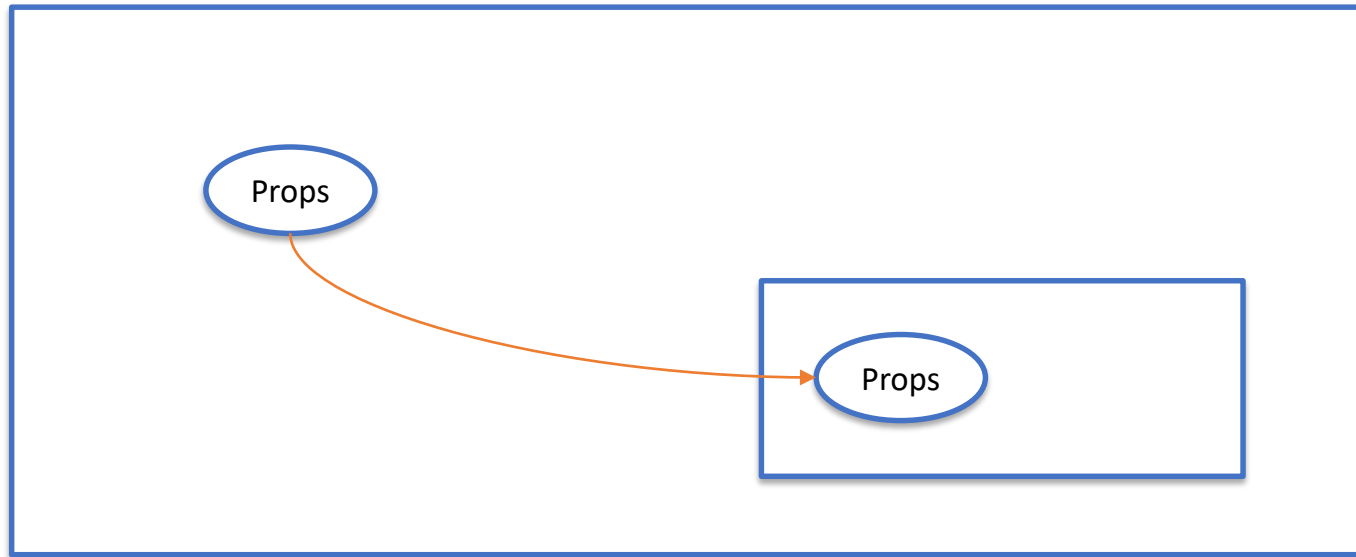
11. default props

12. propTypes

## 9. 프로퍼티 (Props)

## 9. 프로퍼티 (Props)

- 컴포넌트에서 사용할 데이터 중 변경되지 않는 데이터를 처리할 때 사용.
- 일반적으로 부모 컴포넌트에서 자식 컴포넌트로 데이터(함수포함)를 전달할 때 사용된다.
- React에서 모든 데이터의 흐름은 “단 방향” 으로 처리해야 한다.



## 9. 프로퍼티 (Props)

- 부모에서 자식 컴포넌트의 속성에 값을 설정하여 전달한다.
- 문자열 이외의 값을 전달할 때는 반드시 {값} 형식으로 사용한다.

```
import Card from './prop/Card';

function App() {
  return (
    <div>
      <Card name="장민창" age={39} />
    </div>
  );
}

export default App;
```

```
export default class Card extends Component {
  render() {
    const { name, age } = this.props;
    return (
      <div>
        <div>
          name: <span style={{ "font-weight": "bold" }}>{name}</span>
        </div>
        <div>
          age: <span style={{ "font-weight": "bold" }}>{age}</span>
        </div>
      </div>
    );
  }
}
```

## 9. 프로퍼티 (Props)

- 실습

```
function App() {  
  let content = [  
    {"num": 1, "name": "이순신", "age": 20},  
    {"num": 2, "name": "유관순", "age": 30},  
    {"num": 3, "name": "강감찬", "age": 40}  
  ];  
  
  return (  
    <div>  
      <Table content={content} />  
    </div>  
  );  
}
```

번호	이름	나이
----	----	----

1	이순신	20
---	-----	----

2	유관순	30
---	-----	----

3	강감찬	40
---	-----	----

```
▼ App  
  ▼ Table  
    Row key="0"  
    Row key="1"  
    Row key="2"
```

# **10. Component Body Props**



# 10. Component Body Props

- 컴포넌트와 컴포넌트 사이의 Body 값을 Props로 사용 가능

```
return (  
  <div>  
    <Table content={content}>  
      Hello<span>React</span><span>JS</span>  
    </Table>  
  </div>  
)
```

```
export class Table extends Component {  
  render() {  
    const {content, children} = this.props;  
  
    return (  
      <table>  
        <thead> ...  
      </thead>  
      <tbody> ...  
      </tbody>  
      <tfoot>  
        <tr>  
          <td>{children[0]}</td>  
          <td>{children[1]}</td>  
          <td>{children[2]}</td>  
        </tr>  
      </tfoot>  
    </table>  
  );  
}
```

# **11. default props**

# 11. default props

- props 데이터가 필요한 컴포넌트를 사용할 때 props를 전달하지 않았을 경우 기본으로 보여줄 기본 props를 정의
- 컴포넌트 하단에 아래와 같은 형태로 삽입
- `ClassComponentName.defaultProps = {propName: propValue}`

```
export default class Child extends Component {
  render() {
    const {name, age} = this.props;
    return (
      <div>
        <span>{name}</span> / <span>{age}</span>
      </div>
    );
  }
}

Child.defaultProps = {
  name: "아무개",
  age: 20
}
```

```
function App() {
  return (
    <div>
      <Child/>
      <Child name="이순신" age={30}/>
    </div>
  );
}
```

## 12. propTypes

# 12. propTypes

- props데이터는 propTypes로 검사가 가능하다

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

export default class Child extends Component{ ..
}

Child.defaultProps = { ...
}

Child.propTypes = {
  name: PropTypes.string,
  age: PropTypes.number
};
```

# 12. propTypes

- props데이터는 propTypes로 검사가 가능하다

```
function App() {  
  return [  
    <div>  
      <Child  
        boolValue={false}  
        numValue={30}  
        arrayValue={[1,2,3]}  
        objValue={ {name: '홍길동', age: 20} }  
        nodeValue={<h1>노드</h1>}  
        funcValue={() => console.log('func')}  
        oneOfTypeValue={10}  
        oneOfValue='여' />  
      </div>  
    ];  
}
```

```
Child.propTypes = {  
  boolValue: PropTypes.bool,  
  numValue: PropTypes.number,  
  arrayValue: PropTypes.array,  
  objValue: PropTypes.object,  
  nodeValue: PropTypes.node,  
  funcValue: PropTypes.func,  
  oneOfTypeValue: PropTypes.oneOfType([  
    PropTypes.string,  
    PropTypes.number  
  ]),  
  oneOfValue: PropTypes.oneOf(['남', '여'])  
};
```

- 필수값 체크여부는 isRequired 로 할 수 있다.

```
Child.propTypes = {  
  boolValue: PropTypes.bool.isRequired,  
  numValue: PropTypes.number,
```

상호작용

# Event

---

13. React의 Event 처리 방법

14. 'this' in Events

15. 이벤트 기본 동작 방지

16. 하위컴포넌트에 이벤트 전달

## 13. React의 Event 처리 방법



# 13. React의 Event 처리 방법

- 컴포넌트 또는 HTML Element에 이벤트를 부여
  - 마우스 이벤트
  - 키보드 이벤트
  - 전송 이벤트 등등
- 기존 HTML의 이벤트와 React의 이벤트의 차이 (Camel Case)

- HTML

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

- React

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

## 14. 'this' in Events

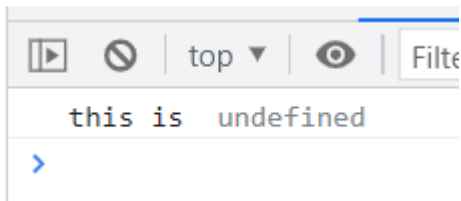
# 14. 'this' in Events

- 이벤트를 할당 할 때 아래처럼 정의할 경우, this 객체는 undefined가 된다.

```
export default class Child extends Component{

  handleClick() {
    console.log("this is ", this);
  }

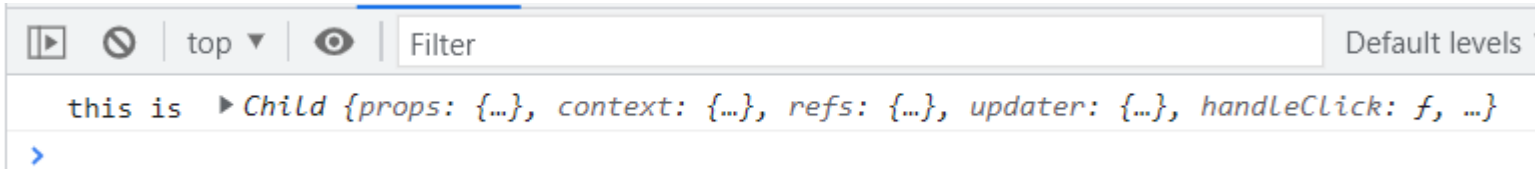
  render() {
    return (
      <div>
        <button onClick={this.handleClick}>Click Me</button>
      </div>
    );
  }
}
```



# 14. 'this' in Events

- 이벤트에서 'this'를 사용하려면 바인딩이 필요하다.

```
export default class Child extends Component{  
  
  constructor(props) {  
    super(props);  
    this.handleClick = this.handleClick.bind(this);  
  }  
  
  handleClick() {  
    console.log("this is ", this);  
  }  
  
  render() {  
    return (  
      <div>  
        <button onClick={this.handleClick}>Click Me</button>  
      </div>  
    );  
  }  
}
```



# 14. 'this' in Events

- 바인딩 없이 사용하려면 Inline Event로 변경한다.

```
export default class Child extends Component{

  handleClick() {
    console.log("this is ", this);
  }

  render() {
    return (
      <div>
        <button onClick={() => this.handleClick()}>Click Me</button>
      </div>
    );
  }
}
```

Debugger interface showing the state of the component:

top | Filter | Default levels ▼

this is  
▶ Child {props: {...}, context: {...}, refs: {...}, updater: {...}, \_reactInternals: FiberNode, ...}

>

## 15. 이벤트 기본 동작 방지

# 15. 이벤트 기본 동작 방지

---

- 기본 이벤트가 할당되어 있는 HTML 엘리먼트들에게 이벤트를 할당해야 할 경우.
  - 예1> HTML Form Element에게 이벤트를 할당한다.
- 예로, Form Element에 Submit 함수의 기본 동작 형태를 변경하려면, 아래와 같이 이벤트 객체에 `preventDefault();` 를 실행시키면 된다.

```
handleClick(event) {  
    event.preventDefault();  
    console.log(this, event);  
}
```

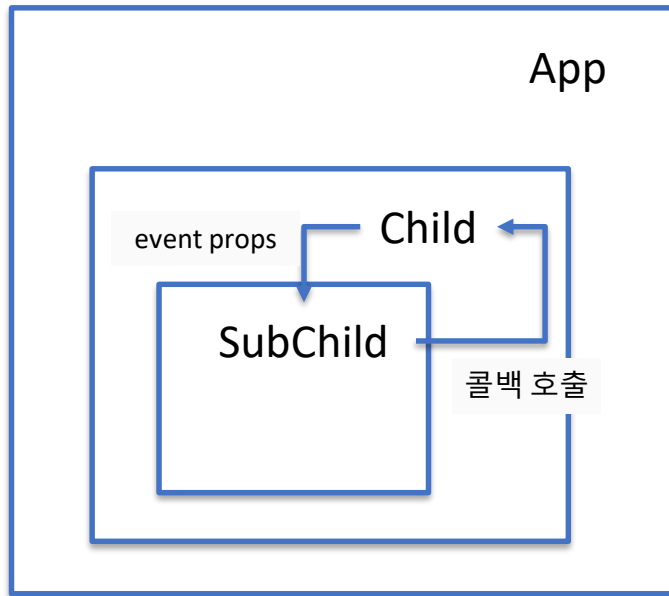
- Form Element의 기본 Submit 이벤트는 무시되며 하위의 코드들이 실행된다.

## **16. 하위컴포넌트에 이벤트 전달**



# 16. 하위컴포넌트에 이벤트 전달

- props를 이용해 데이터 전달을 했던 것 처럼 이벤트 함수도 props로 전달이 가능하다.



# 16. 하위컴포넌트에 이벤트 전달

```
export default class SubChild extends Component {
```

```
  render() {
    const {onEvent} = this.props;

    return (
      <React.Fragment>
        <h1>Child의 콜백 호출</h1>
        <button onClick={() => onEvent(this, "SubChild")}>Click Me!</button>
      </React.Fragment>
    );
  }
}
```

```
export default class Child extends Component{

  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick(component, text) {
    console.log("Component", component);
    console.log("Text", text);
    console.log("This is, this");
  }

  render() {
    return (
      <div>
        <SubChild onEvent={this.handleClick} />
      </div>
    );
  }
}
```

컴포넌트의 데이터

# State

---

17. State 개요

18. 클래스 컴포넌트의 State

19. 함수형 컴포넌트의 State

## 17. State 개요

# 17. State 개요

---

- 컴포넌트가 사용하는 데이터.
  - 서버로부터 데이터를 가져와 화면에 구성
  - **데이터가 변경되면 화면이 변경 됨.**  
**(State가 변경되면 View가 자동으로 갱신됨)**
  - 이러한 데이터를 컴포넌트(화면)의 상태(State) 라고 함.
- \* Props는 부모로 부터 전달되는 불변 데이터. \*

## 18. 클래스 컴포넌트의 State

# 18. 클래스 컴포넌트의 State

- 클래스 컴포넌트
  - 생성자
  - 업데이트
  - 렌더링

```
export default class Child extends Component {  
  
  constructor(props) {  
    super(props);  
    this.handleChange = this.handleChange.bind(this);  
    this.state = {  
      key: ""  
    };  
  }  
  
  handleChange(event) {  
    this.setState({key: event.target.value});  
  }  
  
  render() {  
    return (  
      <div>  
        <div>{this.state.key}</div>  
        <input type="text" onChange={this.handleChange} />  
      </div>  
    );  
  }  
}
```

# 18. 클래스 컴포넌트의 State

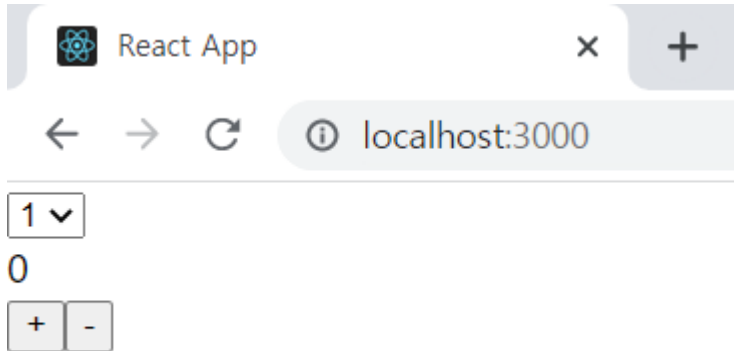
- state의 값을 props를 이용해 수정할 경우 정상적인 업데이트가 실패할 수 있으므로 setState에 state와 props를 동시에 전달한다.

```
increase() {  
  if (this.state.counter < 100) {  
    this.setState((state, props) => ({  
      counter: state.counter + props.step  
    }));  
  }  
}  
  
decrease() {  
  if (this.state.counter > 0) {  
    this.setState((state, props) => ({  
      counter: state.counter - props.step  
    }));  
  }  
}
```



# 18. 클래스 컴포넌트의 State

- (실습) Counter 만들어 보기



# 18. 클래스 컴포넌트의 State

```
export default class Child extends Component{

  constructor(props) {
    super(props);
    this.selectChange = this.selectChange.bind(this);

    this.state = {
      step: 1
    };
  }

  selectChange(event) {
    this.setState({step: parseInt(event.target.value)});
  }

  render() {
    return (
      <div>
        <select onChange={this.selectChange}>
          <option value="1">1</option>
          <option value="2">2</option>
          <option value="3">3</option>
        </select>
        <SubChild step={this.state.step} />
      </div>
    );
  }
}
```

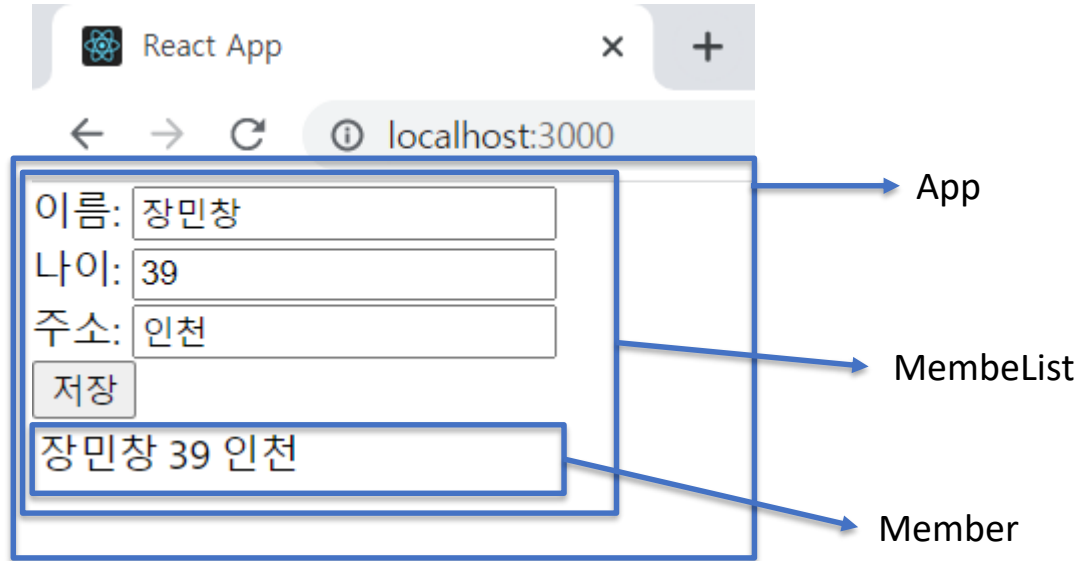
# 18. 클래스 컴포넌트의 State

```
export default class SubChild extends Component {  
  
  constructor(props) {  
    super(props);  
    this.increase = this.increase.bind(this);  
    this.decrease = this.decrease.bind(this);  
  
    this.state = {  
      counter: 0  
    }  
  }  
  
  increase() {  
    if (this.state.counter < 100) {  
      this.setState((state, props) => ({  
        counter: state.counter + props.step  
      }));  
    }  
  }  
}
```

```
  decrease() {  
    if (this.state.counter > 0) {  
      this.setState((state, props) => ({  
        counter: state.counter - props.step  
      }));  
    }  
  }  
  
  render() {  
    return (  
      <React.Fragment>  
        <div>{this.state.counter}</div>  
        <button onClick={this.increase}>+</button>  
        <button onClick={this.decrease}>-</button>  
      </React.Fragment>  
    );  
  }  
}
```

# 18. 클래스 컴포넌트의 State

- (실습) 주소록 만들어보기



# 18. 클래스 컴포넌트의 State

- (실습) 주소록 만들어보기 - MemberList Event

```
constructor(props) {  
  super(props);  
  this.textChange = this.textChange.bind(this);  
  this.save = this.save.bind(this);  
  this.state = {  
    memberData: [],  
    username: "",  
    age: "",  
    address: ""  
  }  
}  
  
textChange(event) {  
  this.setState({  
    [event.target.id]: event.target.value  
  });  
}  
  
save() {  
  let memberList = this.state.memberData;  
  memberList.push({username: this.state.username, age: this.state.age, address: this.state.address});  
  this.setState({  
    memberData: memberList  
  });  
}
```

## 19. 함수형 컴포넌트의 State

# 19. 함수형 컴포넌트의 State

---

- 리액트 훅(Hook) 중 useState()를 통해 상태를 관리한다.

## 7. 함수형 / 클래스

---

- 일반적인 컴포넌트는 Class 형태로 선언.
- 다음과 같은 조건이 있을 경우, 함수형 컴포넌트 사용 가능
  - 1. Lifecycle API를 사용하지 않을 경우
  - 2. State를 사용하지 않을 경우
  - 3. Props만 사용하는 경우
- React Hook
  - 함수형 컴포넌트에서 Lifecycle, State, Props 사용이 가능함.

- `const [변수명, set변수명] = useState(기본값);`

```
import { useState } from "react";  
const [step, setStep] = useState(1);
```

# 19. 함수형 컴포넌트의 State \_ 1

- useState() Hook

```
1  import { useState } from "react";
2
3  export default function Counter() {
4      const [step, setStep] = useState(1);
5
6      function changeStep(event) {
7          setStep(parseInt(event.target.value));
8      }
9
10     return (
11         <div>
12             <select onChange={changeStep}>
13                 <option value="1">1</option>
14                 <option value="2">2</option>
15                 <option value="3">3</option>
16             </select>
17             <Stepper step={step} />
18         </div>
19     );
20 }
```



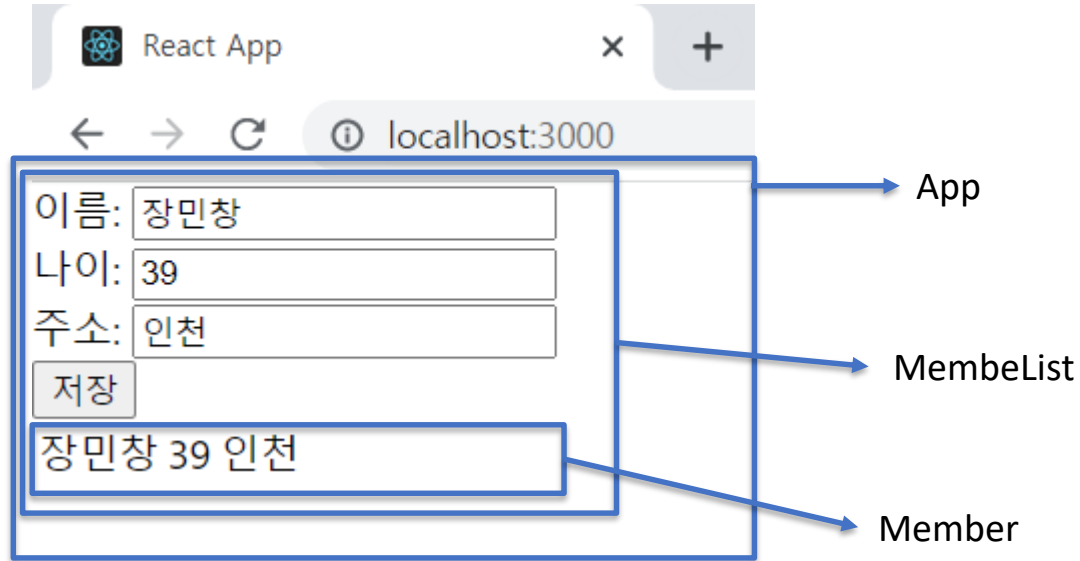
# 19. 함수형 컴포넌트의 State \_ 2

- useState() Hook

```
22  function Steper(props) {
23      const [count, setCount] = useState(0);
24      const step = props.step;
25
26      function increase() {
27          if (count + step < 100) {
28              setCount(count + step);
29          }
30      }
31
32      function decrease() {
33          if (count - step > 0) {
34              setCount(count - step);
35          }
36      }
37
38      return (
39          <>
40              <div>{count}</div>
41              <button onClick={increase}>+</button>
42              <button onClick={decrease}>-</button>
43          </>
44      );
45  }
```

# 19. 함수형 컴포넌트의 State

- (실습) 주소록 만들어보기



# 19. 함수형 컴포넌트의 State

- (실습) 주소록 만들어보기 - MemberList Event

```
const [memberData, setMemberData] = useState([]);
const [name, setName] = useState("");
const [age, setAge] = useState("");
const [address, setAddress] = useState("");
```

```
function changeName(event) {
  setName(event.target.value);
}
```

```
function changeAge(event) {
  setAge(event.target.value);
}
```

```
function changeAddress(event) {
  setAddress(event.target.value);
}
```

```
function save() {
  let data = [...memberData];
  data.push({username: name, age, address});
  setMemberData(data);
}
```

# 컴포넌트 스타일링

---

20. CSS

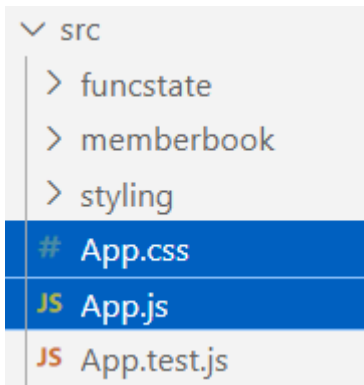
21. Inline-Style

22. Literal Style

# 20. CSS

## 20. CSS

- 일반적인 Web의 경우 공통적인 CSS를 정의하여 사용.
  - CSS 파일이 거대해짐
    - CSS 수정을 위해 많은 시간이 소요됨.
    - CSS 수정 후 Side Effect 발생 가능성이 매우 높음
- React는 하나의 컴포넌트에 해당되는 CSS 파일을 별도로 정의함.
  - CSS 파일이 세분화되어 관리.
    - CSS수정을 위해 비교적 적은 시간이 소요됨.
    - CSS 수정 후 Side Effect 발생 가능성이 비교적 낮음

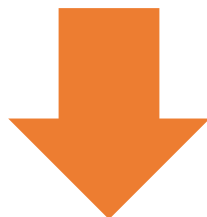


## 20. CSS

src  
 > funcstate  
 > memberbook  
 styling  
 # StyleDiv.css  
 JS StyleDiv.js

```
1  import "../StyleDiv.css";  
2  
3  export function Box(props) {  
4    return (  
5      <div className="box">  
6        {props.letter}  
7      </div>  
8    );  
9  }
```

```
1  div.box {  
2    padding: 10px;  
3    margin: 10px;  
4    background-color: #FFDE00;  
5    color: #333;  
6    display: inline-block;  
7    font-size: 32px;  
8    font-family: monospace;  
9    text-align: center;  
10 }
```



A E I O U

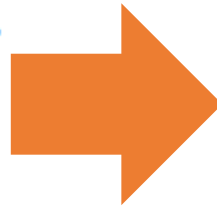
# 21. Inline-Style



# 21. Inline-Style

- 컴포넌트 내부에서 스타일을 정의할 경우, {} 에 정의한다.

```
1 export function InlineBox(props) {  
2   return (  
3     <div className="box" style={{  
4       padding: "10px",  
5       margin: "10px",  
6       backgroundColor: "#FFDE00",  
7       color: "#333",  
8       display: "inline-block",  
9       fontSize: "32px",  
10      fontFamily: "monospace",  
11      textAlign: "center"  
12    }}>  
13      {props.letter}  
14    </div>  
15  );  
16 }
```



# 21. Inline-Style

- 리터럴 정의할 경우, 아래와 같이 props를 사용할 수도 있다.

```
4 function App() {  
5  
6   return (  
7     <>  
8       <InlineBox letter="A" bgColor="#EEE"/>  
9       <InlineBox letter="E" bgColor="#CCC" />  
10      <InlineBox letter="I" bgColor="#AAA" />  
11      <InlineBox letter="O" bgColor="#999" />  
12      <InlineBox letter="U" bgColor="#888" />  
13    </>  
14  );  
15 }
```

```
1 export function InlineBox(props) {  
2   return (  
3     <div className="box" style={{  
4       padding: "10px",  
5       margin: "10px",  
6       backgroundColor: "#FFDE00",  
7       color: "#333",  
8       display: "inline-block",  
9       fontSize: "32px",  
10      fontFamily: "monospace",  
11      textAlign: "center"  
12    }}>  
13      {props.letter}  
14    </div>  
15  );  
16 }
```



# 21. Literal Style

## 22. Literal Style

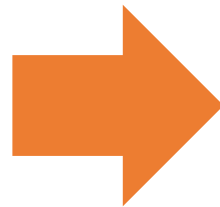
---

- React 의 Inline-Style 객체를 변수로 정의하여 사용할 경우, Literal Style 로 정의할 수 있다.
  - `Inline-Style == LiteralStyle`
  - 차이는 변수로 정의했는지, 인라인으로 정의했는지의 차이

## 22. Literal Style

- React 의 Inline-Style 객체를 변수로 정의하여 사용할 경우, Literal Style 로 정의할 수 있다.
  - Inline-Style == LiteralStyle
  - 변수로 정의했는지, 인라인으로 정의했는지의 차이

```
1  export function InlineBox(props) {  
2      const style = {  
3          padding: "10px",  
4          margin: "10px",  
5          backgroundColor: props.bgColor,  
6          color: "#333",  
7          display: "inline-block",  
8          fontSize: "50px",  
9          fontFamily: "monospace",  
10         textAlign: "center"  
11     };  
12  
13     return (  
14         <div className="box" style={style}>  
15             {props.letter}  
16         </div>  
17     );  
18 }
```



# 컴포넌트 라이프 사이클

---

23. 라이프 사이클 개요

24. 컴포넌트의 라이프 사이클

## 23. 라이프 사이클 개요

## 23. 라이프 사이클 개요

---

- 클래스 컴포넌트는 여러 종류의 라이프사이클 메소드를 제공한다.
- React에서 제공하는 라이프사이클 메소드를 재정의 해 특정 시점에 코드가 실행되도록 할 수 있다.
  - ~~함수형 컴포넌트는 사용할 수 없다.~~
  - React Hook으로 간접 사용할 수 있게 되었다.
    - **함수형 컴포넌트는 라이프사이클을 사용할 수 없다.**
- React가 제어할 수 없는 부작용(Side effect)를 처리 하기 위해 사용됨.
  - DOM Handling (실제 문서(Document)에 접근 및 조작)
  - 네트워크 통신 (비동기 통신 요청 및 응답 처리)
    - Fetch API 또는 axios 라이브러리
  - 이벤트 핸들링 작업에서 구독 및 취소 작업
    - 컴포넌트 생성시점에 이벤트 구독했으면, 제거시점에 이벤트를 취소해야함.



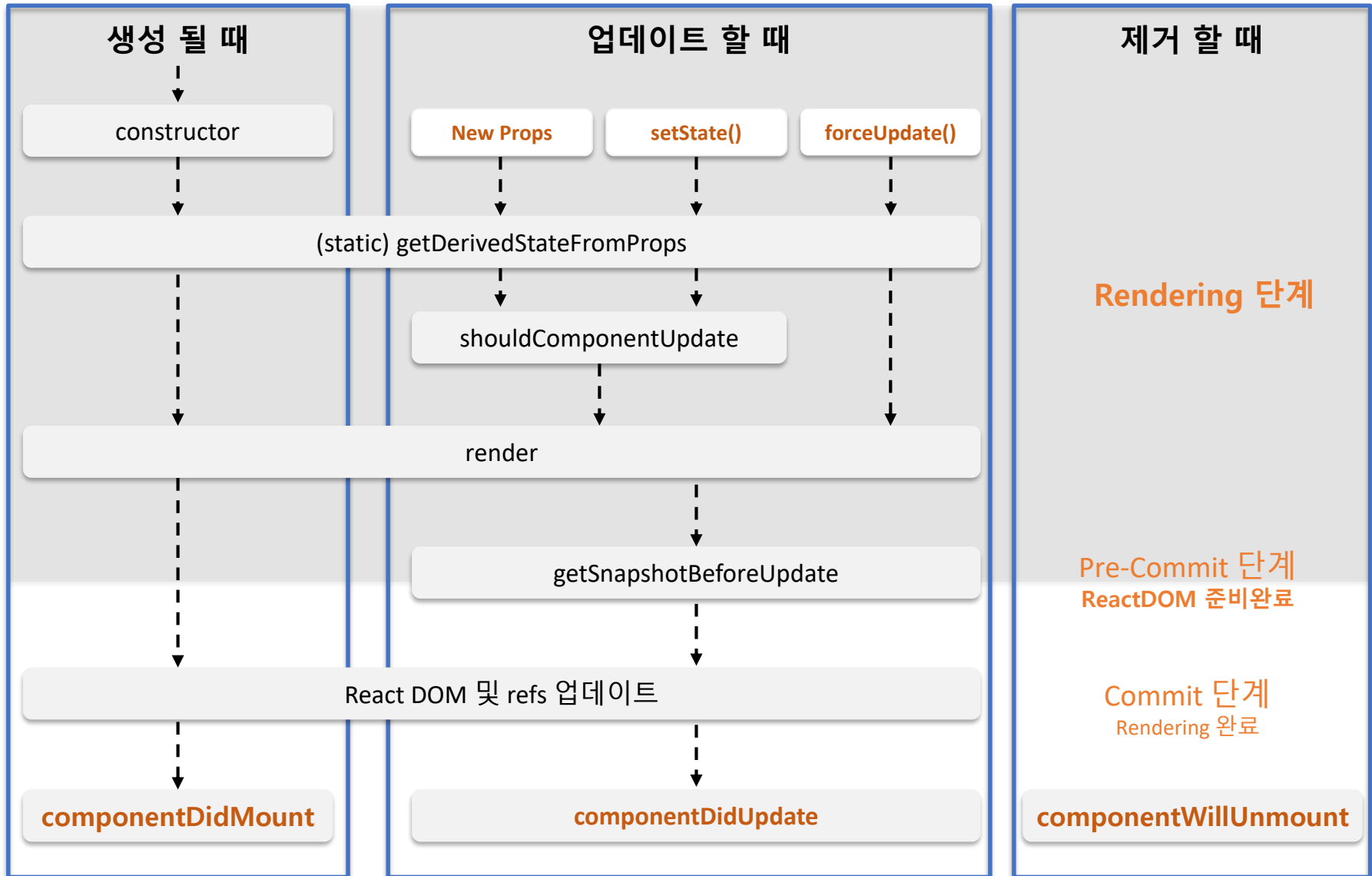
## 23. 라이프 사이클 개요

---

- React 개발시 라이프사이클을 사용할 경우
  - Props 혹은 State가 변경되었을 때
  - 컴포넌트가 Re-Rendering을 한다.
  - Re-Rendering된 컴포넌트에 자식 컴포넌트가 있을 때
  - 자식 컴포넌트도 Rerendering을 한다.
- setState()를 사용하지 않고 state를 변경해 this.forceUpdate로 컴포넌트 강제 Re-Rendering을 할 때

## 24. 컴포넌트의 라이프 사이클

# 24. 컴포넌트의 라이프 사이클



# 24. 컴포넌트의 라이프 사이클

---

- 1. 렌더링 전/후 (마운트 상태)
  - constructor
    - 새로운 객체가 생성될 때 **마다 실행**
    - this.props 초기화 가능, this.setState 또는 Ajax 등의 처리 불가
  - static getDerivedStateFromProps
    - 컴포넌트 초기화 또는 **새로운 props를 받았을 때 실행.**
    - this키워드는 사용할 수 없음.
  - render
    - 화면에 컴포넌트를 작성
    - this.props와 this.state를 분석 후 화면에 노출
    - state를 변경할 수 없으며, 브라우저 API에 직접 접근 불가
  - **componentDidMount**
    - **컴포넌트 렌더링이 완료되었을 때 실행**
    - DOM에 접근이 가능하며, Ajax 및 데이터 로직 처리에 적합

## 24. 컴포넌트의 라이프 사이클

---

- 2. 수정 전/후 (업데이트 상태)
  - `shouldComponentUpdate (nextProps, nextState) {}`
    - **state가 변경 될 경우** `shouldComponentUpdate` 부터 라이프사이클 진행.
  - `getSnapshotBeforeUpdate (prevProps, prevState) {}`
    - **render 메서드 호출 후 DOM에 변화를 반영하기 바로 전에 호출**
    - 이 메서드에서 리턴된 값을 `componentDidUpdate` 메서드의 세 번째 파라미터(`snapshot`) 에서 받을 수 있다.
    - 주로 업데이트하기 직전의 값을 참고할 때 사용된다.
  - **`componentDidUpdate (prevProps, prevState, snapshot) {}`**
    - 모든 re-rendering이 완료된 후 호출
    - DOM 처리 가능
    - `prevProps, prevState` 값을 이용해 컴포넌트의 이전 데이터에 접근할 수 있다.

컴포넌트를 위한

# React Hooks

---

25. useState

26. useRef

27. useEffect

28. useMemo

29. useCallback

30. React.memo

31. useReducer

32. useContext

함수형 컴포넌트의 상태 제어

## 25. useState

# 25. useState

---

- State 값 관리를 위한 Hook
  - 함수형 컴포넌트로 state 관리 가능 (React 16.8 에서 Hook 도입)
- `const [number, setNumber] = useState(0);`
  - `number`: state의 이름
  - `setNumber`: `number` state 값 변경함수
    - 호출 후 값이 변경되었다면 컴포넌트를 Re-Rendering한다.
  - `0`: `number` state의 기본 값



Element 참조

## 26. useRef

## 26. useRef

---

- 특정 DOM에 접근하기 위한 Hook
  - jQuery는 브라우저의 DOM을 직접제어함.
    - id, class, data 등으로 제어
    - 속도가 느림.
- React는 JSX 문법을 통해 DOM을 생성함.
  - 가상DOM으로 화면을 제어함
    - id, class, data 등으로 제어할 수 없음.
    - 단, 속도가 매우 빠름
  - 가상DOM을 제어하기 위해 useRef Hook이 필요.

## 26. useRef

```
1  import { useRef } from "react";
2
3  export default function RefComponent() {
4      const nameRef = useRef();
5
6      function click() {
7          var name = nameRef.current;
8          if (name.value == "") {
9              alert("이름을 입력하세요!");
10             name.focus();
11         }
12     }
13
14     return (
15         <div>
16             <input type="text" ref={nameRef}/>
17             <button onClick={click}>클릭!</button>
18         </div>
19     );
20
21 }
```

## 26. useRef (forwardRef)

---

- ref 를 하위 컴포넌트로 보내려면 forwardRef를 사용한다.
- 부모컴포넌트에서 정의한 ref를 하위 컴포넌트의 props 로 전달한다.
- `<ChildComponent ref={inputRef} />`
- 하위 컴포넌트의 extend 문장에 forwardRef로 감싼다.
- `export forwardRef(ChildComponent)`
- 그러면 하위 컴포넌트는 두 개의 인자를 받을 수 있다.
- `function ChildComponent(props, ref) { ... }`

함수형 컴포넌트의 Re-Rendering 제어

## 27. useEffect

# 27. useEffect

---

- React의 주요역할이 아닌 모든 작업을 수행하는 Hook
- React의 주요 역할
  - UI Rendering, 사용자 입력 처리, JSX 렌더링
- React의 주요 역할을 제외한 나머지는 Side Effect 라고 한다.
  - Back-end Server RESTFul API 요청
  - Timer 설정 등
- 클래스 컴포넌트의 라이프 사이클에 대응하기 위해 도입됨.
  - 컴포넌트 마운트 / 언마운트 / 업데이트 되었을 때 수행
  - 혹은 특정 props 혹은 state가 변경될 때 특정 작업 처리 가능

## 27. useEffect

---

- `useEffect(param1, param2)`
  - `param1`: 수행할 코드
  - `param2`: 의존 값
    - 파라미터가 없을 시 컴포넌트 업데이트 시 매번 실행
    - `[]` (빈배열) 전달 시 컴포넌트 최초 마운트 시 1회만 실행
    - 특정 `state` 값 추가 시, `state`가 변경될 때 마다 실행

# 27. useEffect

```
export default function EffectComponent() {  
  const [memberData, setMemberData] = useState([]);  
  const [name, setName] = useState("");  
  const [age, setAge] = useState("");  
  const [address, setAddress] = useState("");  
  
  useEffect(function() {  
    console.log("State가 변경될 때마다 매번 실행");  
  });  
  
  useEffect(function() {  
    console.log("처음 한번만 실행");  
  }, []);  
  
  useEffect(function() {  
    console.log("이름 state가 변경될 때마다 실행");  
  }, [name]);  
  
  useEffect(function() {  
    console.log("나이 또는 주소 state가 변경될 때마다 실행");  
  }, [age, address]);  
  
  useEffect(function() {  
    console.log("memberData state가 변경될 때마다 실행");  
  }, [memberData]);  
}
```

State가 변경될 때마다 매번 실행

처음 한번만 실행

이름 state가 변경될 때마다 실행

나이 또는 주소 state가 변경될 때마다 실행

memberData state가 변경될 때마다 실행

State가 변경될 때마다 매번 실행

이름 state가 변경될 때마다 실행

State가 변경될 때마다 매번 실행

나이 또는 주소 state가 변경될 때마다 실행

State가 변경될 때마다 매번 실행

나이 또는 주소 state가 변경될 때마다 실행

State가 변경될 때마다 매번 실행

memberData state가 변경될 때마다 실행



컴포넌트 최적화를 위한 데이터 캐시

## 28. useMemo

## 28. useMemo

---

- 데이터 캐시를 위한 Hook
  - 함수 컴포넌트 내의 지역변수는 컴포넌트 리렌더링 후 초기화 됨.
- 함수컴포넌트가 리렌더링 된 후 초기화되면?
  - 함수 내의 함수를 다시 호출 하게 됨
  - 만약 오래걸리는 함수를 호출한다면, 렌더링하는데 오랜 시간이 걸림

## 28. useMemo

- 함수가 처음 렌더링 되거나 리렌더링 될 경우
  - 항상 calc 함수를 재호출 하게 됨.
- state의 값이 변경될 경우에도 함수가 리렌더링 되므로
  - 항상 calc 함수를 재호출 하게 됨.

```
1  import { useState } from "react";
2
3  export default function MemoComponent() {
4    const [number, setNumber] = useState(1);
5    const [fastNumber, setFastNumber] = useState(1);
6
7    const calc = function(number) {
8      for (var i = 0; i < 9999999999; i++) {}
9      return number + 10000;
10   }
11   const fastestCalc = function(number) {
12     return number + 10000;
13   }
14
15   const result = calc(number);
16   const fastestResult = fastestCalc(fastNumber);
```

## 28. useMemo

- 빠른 계산기의 값을 수정하더라도 항상 오래 걸리게 된다.
  - 오래 걸리는 계산기의 값을 캐시에 저장해 둔다면?

```
17
18     return (
19         <div>
20             <h3>오래 걸리는 계산기</h3>
21             <input type="number"
22                 |         defaultValue={number}
23                 |         onChange={(e) => {setNumber(parseInt(e.target.value))}} />
24             + <span>10000</span> +
25             <span>{result}</span>
26
27             <h3>빠른 계산기</h3>
28             <input type="number"
29                 |         defaultValue={fastNumber}
30                 |         onChange={(e) => {setFastNumber(parseInt(e.target.value))}} />
31             + <span>10000</span> +
32             <span>{fastestResult}</span>
33         </div>
34     );
35 }
```

## 28. useMemo

---

```
15 |     const result = useMemo(() => calc(number), [number]);  
16 |     const fastestResult = fastestCalc(fastNumber);
```

- 15번 라인은 처음 계산된 값을 저장하고 있음.
  - number state의 값이 변경될 때에만 콜백함수가 수행되어 결과를 캐싱한다.
  - 컴포넌트가 리렌더링 될 때에도 캐싱된 데이터를 가져온다.
- useMemo(param1, param2)
  - param1: 콜백함수
  - param2: 콜백함수가 수행될 의존 값 (배열)
    - 특정 state 값 추가 시, state가 변경될 때 마다 실행

## 28. useMemo

---

- useMemo에 Literal Object를 넣을 경우, 항상 콜백함수를 실행한다.
  - Primitive Type 일 때엔 값을 할당하며
  - Reference Type 일 때엔 메모리의 주소를 할당하기 때문.
  - **useEffect 도 동일하다.**
- 같은 값을 가지는 Literal Object 라도 메모리의 주소가 다르기 때문에 State가 항상 다르다고 판단한다.
  - 따라서 의존 배열에는 Literal Object(Non-State)를 넣지 않아야 하며
  - 항상 State의 값을 넣어야 한다.
  - **useEffect 도 동일하다.**

컴포넌트 최적화를 위한 함수 캐시

## 29. useCallback

# 29. useCallback

- 함수 자체를 캐싱하는 Hook
  - 함수 컴포넌트 내의 지역변수는 컴포넌트 리렌더링 후 초기화 됨.

- 함수가 리렌더링  
되면 새로운  
함수가 할당된다

```
1  import { useState } from "react";
2
3  export default function CallbackComponent() {
4    const [number, setNumber] = useState(0);
5
6    const someFunction = () => {
7      console.log(`someFunc: number: ${number}`);
8      return;
9    };
10
11    return (
12      <div>
13        <input
14          type="number"
15          value={number}
16          onChange={(e) => setNumber(e.target.value)} />
17        <br/>
18        <button onClick={someFunction}>Call someFunc</button>
19      </div>
20    )
21  }
```



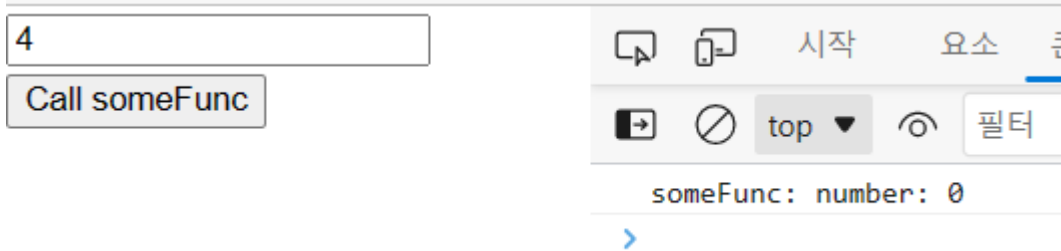
## 29. useCallback

- 함수가 재할당되지 않도록 하려면 useCallback을 사용한다.

```
1  import { useCallback, useState } from "react";
2
3  export default function CallbackComponent() {
4    const [number, setNumber] = useState(0);
5
6    const someFunction = useCallback(() => {
7      console.log(`someFunc: number: ${number}`);
8      return;
9    }, []);
10
11    return (
12      <div>
13        <input
14          type="number"
15          value={number}
16          onChange={(e) => setNumber(e.target.value)} />
17        <br/>
18        <button onClick={someFunction}>Call someFunc</button>
19      </div>
20    )
21  }
```

## 29. useCallback

- 그러나, 문제가 있는 듯한 현상이 발생한다.



- 현재 state 값은 4인데, 콘솔엔 0으로 출력되고 있는 문제.
- 버그인듯 하지만 캐싱이 올바르게 잘 되었기 때문에 0으로 출력됨.

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

- number의 값이 0일 때 만들어진 함수이므로 항상 0을 출력한다.

## 29. useCallback

---

- 이제는 number가 변경될 때에만 함수가 변경되도록 한다.

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

- number를 제외한 다른 state가 갱신되더라도 someFunction은 변경되지 않는다.

컴포넌트 리렌더링 방지

## 30. React.memo

# 30. React.memo

---

- 부모 Component가 렌더링되면 항상 자식 Component도 렌더링된다.
- Component가 불필요하게 Re-Rendering 될 때 사용할 수 있는 Hook
  - 컴포넌트가 같은 Props로 자주 렌더링 될 때
  - 컴포넌트가 렌더링이 될 때마다 복잡한 로직을 처리해야 할 때
  - 그 외에서는 사용하지 않는 것이 좋다.
    - 캐싱에 필요한 메모리가 더 필요해지기 때문.
- 컴포넌트가 받는 **Props가 변경될 때**에만 렌더링 되도록 할 수 있다.
  - State가 변경되면 Re-Rendering이 된다.
  - context가 변경되면 Re-Rendering이 된다.

# 30. React.memo

---

- Props가 변경될 때에만 Re-Rendering 을 하려면
  - `export [default] React.memo(FunctionalComponent);`
- 부모 컴포넌트가 전달하는 Props가 Object (함수포함) 라면 메모리 주소를 전달하게 되므로 Prop Check가 실패한다.
  - 즉, 다른 Prop 이므로 렌더링을 수행한다.
  - Props가 함수, 객체 이 Reference Type일 경우
    - 타입에 따라 `useMemo`, `useCallback`을 사용하면 Re-Rendering을 방지할 수 있다.

상태를 관리하기 위한 또 다른 방법

## 31. useReducer

# 31. useReducer

---

- useState 처럼 state를 생성하고 관리할 수 있도록 하는 hook
  - 복잡한 형태의 객체를 상태로 관리할 때 유용.
- useReducer를 사용하기 위한 기본 지식 3가지
  - Reducer
    - state를 대신 update하는 역할
  - Dispatch
    - Reducer에게 update를 요청하는 역할
  - Action
    - Dispatch에 전달할 내용
      - 예> 예금, 송금, 출금 등



# 31. useReducer

---

- Dispatch(Action) ➔ Reducer(State, Action) ➔ State의 값을 Action으로 업데이트
- ```
const reducer = function(state, action) {  
    // state = money (reducer의 state)  
    // action = dispatch에서 전달한 객체 (type, payload)  
    return state 계산값; // money state의 값이 갱신된다.  
}
```
- ```
const [money, dispatch] = useReducer(reducer, 0);
```

  - money: state
  - dispatch: reducer를 호출
  - reducer: Reducer 객체
  - 0: money의 기본 값
- ```
dispatch({type: "deposit", payload: otherState});
```

  - dispatch를 호출하면 reducer가 실행된다.

# 31. useReducer

---

- useReducer 간단 실습 (개념익히기)
- Reducer 생성

```
3  const reducer = function(state, action) {  
4      console.log("Reducer", "State =>", state, "Action =>", action);  
5      if (action.type == "power") {  
6          return action.payload * action.payload;  
7      }  
8      else if (action.type == "mod") {  
9          return action.payload % 2;  
10     }  
11     else {  
12         return state;  
13     }  
14 };
```

# 31. useReducer

- useReducer 간단 실습 (개념 익히기)
- useReducer 생성

```
16 export default function ReducerComponent() {
17   const [number, setNumber] = useState(1);
18   const [result, dispatch] = useReducer(reducer, 0);
19
20   return (
21     <>
22       <h1>계산기</h1>
23       <h3>결과: {result}</h3>
24       <input type="number"
25         value={number}
26         onChange={(e) => setNumber(parseInt(e.target.value))}/>
27       <button onClick={() => dispatch({type: "power", payload: number})}>제곱구하기</button>
28       <button onClick={() => dispatch({type: "mod", payload: number})}>나머지구하기</button>
29     </>
30   );
31 }
```

# 31. useReducer

- useReducer로 객체 state 관리하기
- ./reducers/TodoReducer.js \_ 1

```
1  export const todoReducer = function(state, action) {
2      const type = action.type;
3      if (type === "add-item") {
4          const newTodoItem = {
5              id: Date.now(),
6              item: action.payload,
7              isComplete: false
8          };
9          return {
10             count: state.count + 1,
11             todos: [...state.todos, newTodoItem],
12             completeTodoCount: state.todos.filter((todo) => todo.isComplete).length
13         }
14     }
```

# 31. useReducer

- ./reducers/ToDoReducer.js \_ 2

```
15     else if (type === "delete-item") {
16         const items = {
17             count: state.count - 1,
18             todos: state.todos.filter((todo) => todo.id !== action.payload)
19         };
20         return {...items, completeTodoCount: items.todos.filter((todo) => todo.isComplete).length}
21     }
22     else if (type === "complete") {
23         const items = {
24             count: state.count,
25             todos: state.todos.map((todo) => {
26                 if (todo.id === action.payload)
27                     return {...todo, isComplete: !todo.isComplete};
28                 return todo;
29             })
30         };
31         return {...items, completeTodoCount: items.todos.filter((todo) => todo.isComplete).length}
32     }
33     else
34         return state;
35 }
```

# 31. useReducer

---

- ./ComplexReducerHook.js \_ 1

```
1  import { useReducer, useState } from "react";
2  import { todoReducer } from "../reducers/ToDoReducer";
3
4  export default function ComplexReducerHook() {
5      const [todo, setTodo] = useState("");
6      const [todoItems, dispatch] = useReducer(todoReducer, {
7          count: 0,
8          completeTodoCount: 0,
9          todos: []
10     });
11 }
```

# 31. useReducer

- ./ComplexReducerHook.js \_ 2

```
12     return (  
13         <>  
14             <h1>Todos</h1>  
15             <div>총 {todoItems.count} 건</div>  
16             <div>총 {todoItems.completeTodoCount} 건 완료</div>  
17             <div>  
18                 <input type="text"  
19                     value={todo}  
20                     onChange={(e) => setTodo(e.target.value)}/>  
21                 &nbsp;   <button onClick={() => dispatch({type: "add-item", payload: todo})}>등록</button>  
22             </div>  
23             {todoItems.todos.map((todo) =>  
24                 <TodoItem key={todo.id}  
25                     id={todo.id}  
26                     item={todo.item}  
27                     isComplete={todo.isComplete}  
28                     dispatch={dispatch} />)}  
29             </>  
30         </>  
31     );  
32 }
```

# 31. useReducer

- ./ComplexReducerHook.js \_ 3

```
33
34 function TodoItem({id, item, isComplete, dispatch}) {
35     return (
36         <div>
37             <span style={{
38                 color: isComplete ? "gray": "black",
39                 textDecoration: isComplete ? "line-through": "none"
40             }} onClick={() => dispatch({type: "complete", payload: id})}>{item}</span>
41             &nbsp;
42             <button onClick={() => dispatch({type: "delete-item", payload: id})}>삭제</button>
43         </div>
44     );
45 }
```



전역 공유

## 32. useContext

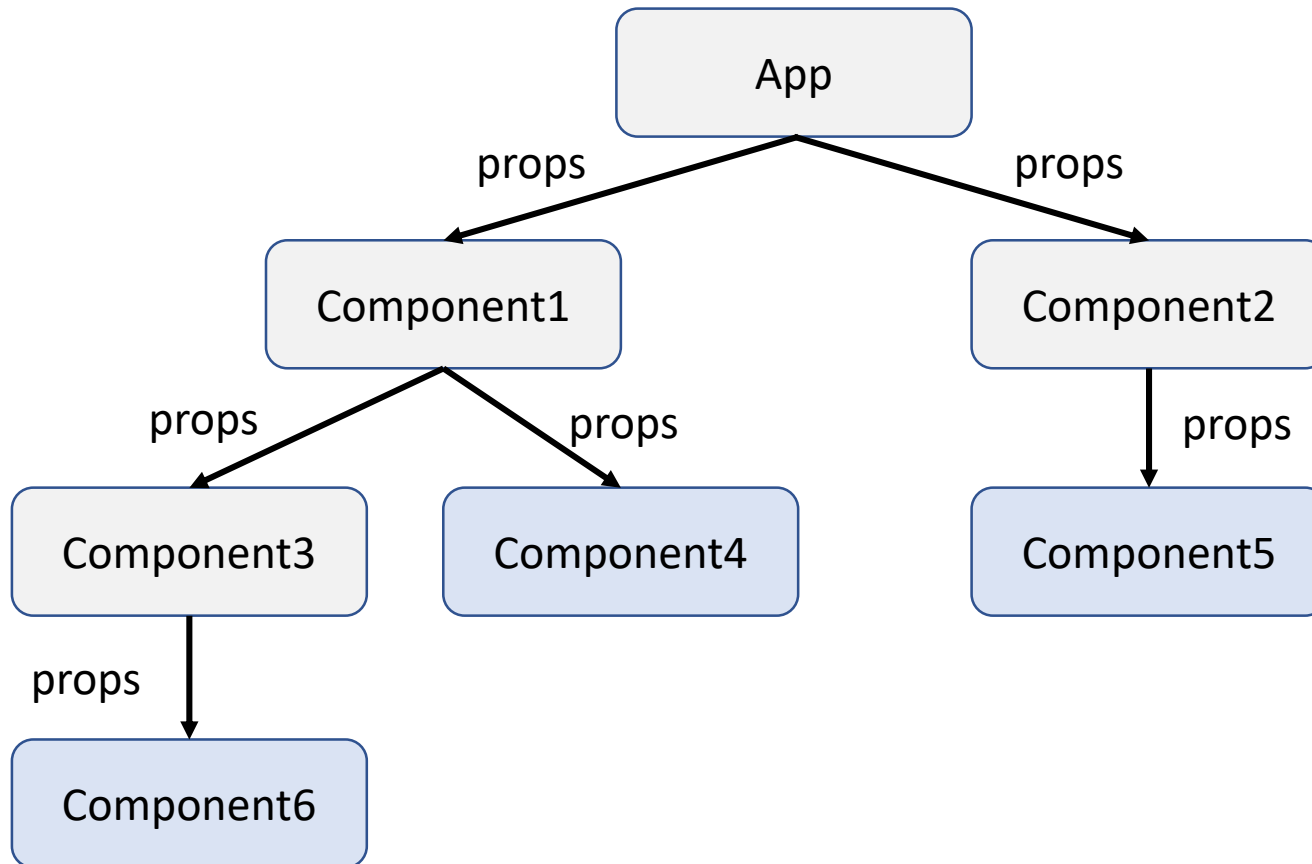
## 32. useContext

---

- 전역으로 사용될 Props를 여러 컴포넌트에서 공유하여 사용할 수 있도록 하는 Hook
  - 예> 로그인 데이터, CSS테마, 언어셋 등
- 이런 데이터들을 모든 컴포넌트가 사용하려면 모든 컴포넌트에 동일한 Props가 전달되어야 함.
  - => Prop Drilling (Prop을 모든 하위 컴포넌트들에게 전달해주는 것)
- 여러 컴포넌트가 Context를 구독하는 형태.
  - Context를 사용하면 컴포넌트를 재사용하기 어려워 질 수 있음.
  - Prop Drilling을 피하기 위해 Context를 사용할 목적이라면 컴포넌트 합성을 사용하는것이 가장 좋은 방법일 수 있다.  
(React Document)

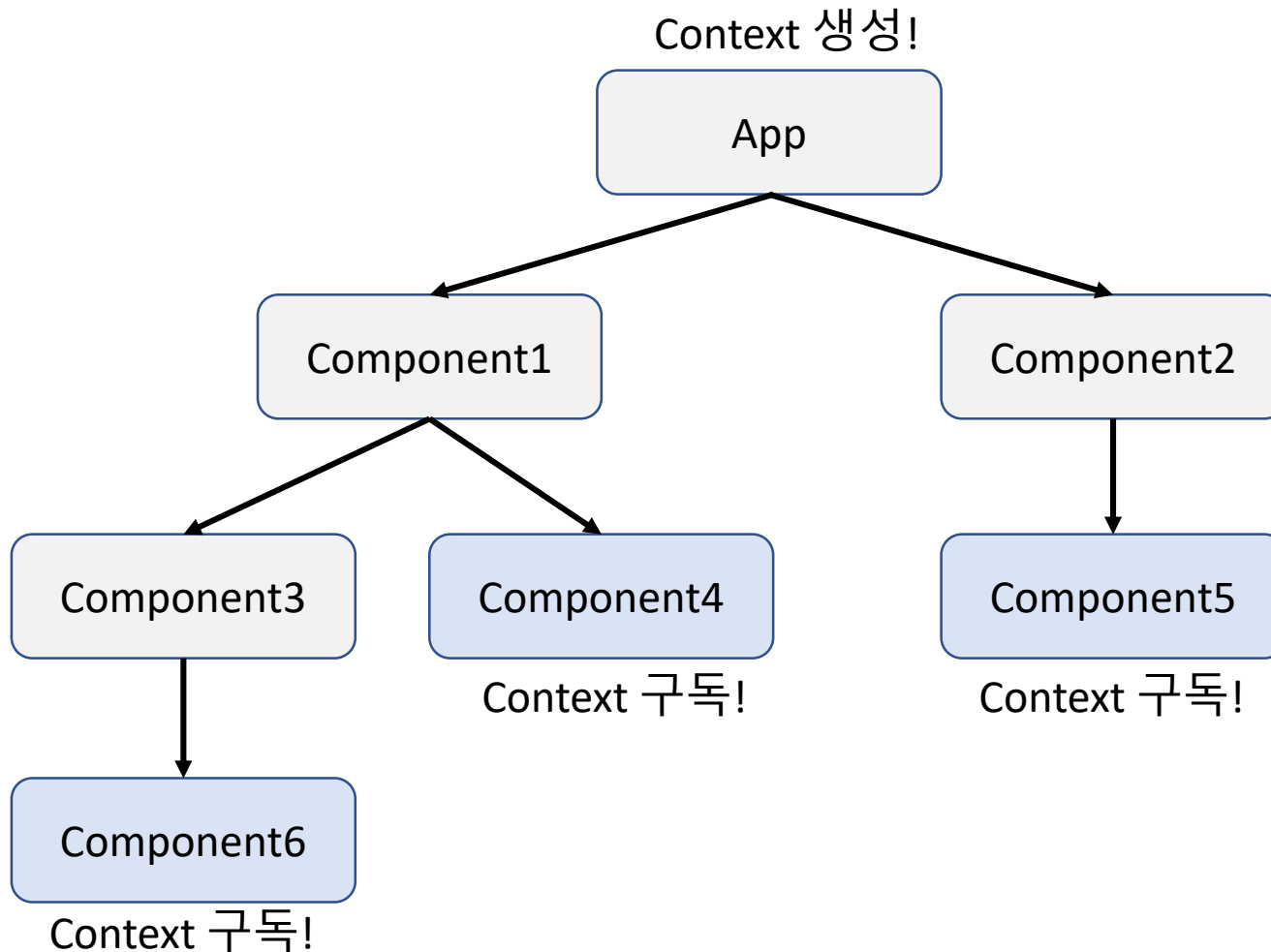
## 32. useContext

- App 에서 Component4, Component5, Component5 로 LoginData Props를 전달하려면..
- Props를 모두 전달해야만 한다. 만약 컴포넌트가 수십개라면?



## 32. useContext

- 같은 상황을 Context로 사용한다면?
- Props를 전달하기 위한 복잡한 과정이 사라진다.



## 32. useContext

- 1. Context 생성 (./contexts/UserContext.js)

```
1 import { createContext } from "react";
2
3 export const UserContext = createContext(null);
```

- 2. UserContext를 사용할 컴포넌트 상위에 ContextProvider 감싸기

```
5 function App() {
6
7   const user = {
8     userId: "abcd",
9     name: "장민창"
10  };
11
12  return (
13    <>
14      <UserContext.Provider value={user}>
15        <CallbackComponent />
16      </UserContext.Provider>
17    </>
18  );
19 }
```

## 32. useContext

- 3. useContext가 필요한 컴포넌트에서 useContext(UserContext) 사용

```
4  export default function CallbackComponent() {
5      const [number, setNumber] = useState(0);
6
7      const someFunction = useCallback(() => {
8          console.log(`someFunc: number: ${number}`);
9          return;
10     }, [number]);
11
12     const {userId, name} = useContext(UserContext);
13
14     return (
15         <div>
16             <input
17                 type="number"
18                 value={number}
19                 onChange={(e) => setNumber(e.target.value)} />
20             <br/>
21             <button onClick={someFunction}>Call someFunc</button>
22             <div>
23                 <span>UserID: {userId}</span>
24                 <span>Name: {name}</span>
25             </div>
26         </div>
27     )
28 }
```

RESTful API Service

# Axios

---

## 33. Axios

서버와 통신을 위한

## **33. Axios**



# 33. Axios

---

- Javascript 환경에서 사용가능한 비동기 통신 방법
  - 1. XMLHttpRequest (전통적 방식)
  - 2. Promise (ECMA6)
  - 3. window.fetch (Promise 기반의 Browser API)
  - 4. axios
    - <https://github.com/axios/axios>
- ECMA6 부터 지원되는 Promise 기반의 비동기 통신 지원
  - HTTP Method 별 API가 제공됨.

# 33. Axios

- react-ui 폴더에서 아래 명령어 입력
  - > npm install axios

```
C:\reactProject\react-ui>npm install axios
added 3 packages, and audited 1474 packages in 5s
231 packages are looking for funding
  run `npm fund` for details
6 high severity vulnerabilities
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
C:\reactProject\react-ui>
```

```
"dependencies": {
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^1.2.5",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-scripts": "5.0.1",
  "web-vitals": "^2.1.4"
},
```

# 33. Axios

---

- config

## Creating an instance

You can create a new instance of axios with a custom config.

`axios.create([config])`

```
const instance = axios.create({
  baseURL: 'https://some-domain.com/api/',
  timeout: 1000,
  headers: {'X-Custom-Header': 'foobar'}
});
```

# 33. Axios

- Sample API Server
  - <https://jsonplaceholder.typicode.com/>

JSONPlaceholder

[Guide](#) [Sponsor this project](#) [Blog](#) [My JSON Server](#)

## {JSON} Placeholder

### Resources

JSONPlaceholder comes with a set of 6 common resources:

|                           |              |
|---------------------------|--------------|
| <a href="#">/posts</a>    | 100 posts    |
| <a href="#">/comments</a> | 500 comments |
| <a href="#">/albums</a>   | 100 albums   |
| <a href="#">/photos</a>   | 5000 photos  |
| <a href="#">/todos</a>    | 200 todos    |
| <a href="#">/users</a>    | 10 users     |

### Routes

All HTTP methods are supported. You can use http or https for your requests.

|        |                                    |
|--------|------------------------------------|
| GET    | <a href="#">/posts</a>             |
| GET    | <a href="#">/posts/1</a>           |
| GET    | <a href="#">/posts/1/comments</a>  |
| GET    | <a href="#">/comments?postId=1</a> |
| POST   | <a href="#">/posts</a>             |
| PUT    | <a href="#">/posts/1</a>           |
| PATCH  | <a href="#">/posts/1</a>           |
| DELETE | <a href="#">/posts/1</a>           |

# 33. Axios

- get (./api/Api.js)
  - remove 동일

```
constructor(props) {  
  super(props);  
  this.state = {  
    error: null,  
    isLoading: false,  
    items: []  
  };  
}
```

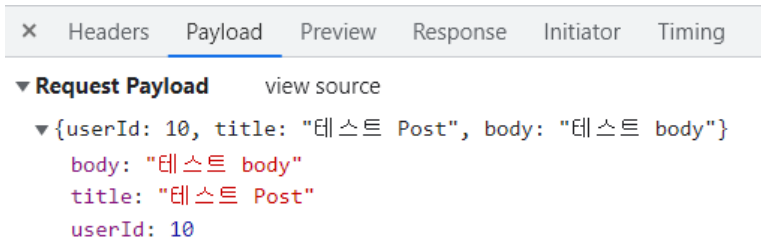
- Leanne Graham Sincere@april.biz
- Ervin Howell Shanna@melissa.tv
- Clementine Bauch Nathan@yesenia.net
- Patricia Lebsack Julianne.OConner@kory.org
- Chelsey Dietrich Lucio\_Hettinger@annie.ca
- Mrs. Dennis Schulist Karley\_Dach@jasper.info
- Kurtis Weissnat Telly.Hoeger@billy.biz
- Nicholas Runolfsdottir V Sherwood@rosamond.me
- Glenna Reichert Chaim\_McDermott@dana.io
- Clementina DuBuque Rey.Padberg@karina.biz

```
/**  
 * 렌더링 즉시  
 * API 호출할 경우 componentDidMount에서 한다.  
 */  
componentDidMount() {  
  axios.get("https://jsonplaceholder.typicode.com/users")  
    .then((result) => {  
      console.log(result);  
      this.setState({  
        ...this.state,  
        isLoading: true,  
        items: result.data  
      });  
    })  
    .catch((error) => {  
      this.setState({  
        ...this.state,  
        isLoading: true,  
        error  
      });  
    });  
});  
}
```

```
render() {  
  const { error, isLoading, items } = this.state;  
  
  if (error) {  
    return <div>Error: {error.message}</div>  
  }  
  else if (!isLoading) {  
    return <div>Loading...</div>  
  }  
  else {  
    return (  
      <ul>  
        {items.map(item => (  
          <li key={item.id}>  
            {item.name} {item.email}  
          </li>  
        ))}  
      </ul>  
    )  
  }  
}
```

# 33. Axios

- post (./api/Api.js) – (put 동일)
  - Request Data를 Http Request Body에 넣어 전송한다.

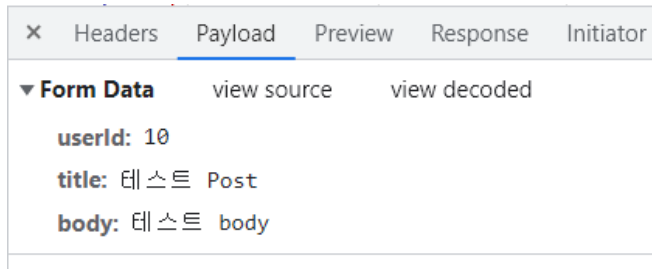


```
/**
 * 렌더링 즉시
 * API 호출할 경우 componentDidMount에서 한다.
 */
componentDidMount() {
  //this.axiosGet();
  axios.post("https://jsonplaceholder.typicode.com/posts", {
    userId: 10,
    title: "테스트 Post",
    body: "테스트 body"
  })
  .then(result => {
    console.log(result);
  })
  .catch(error => {
    console.log(error);
  });
}
```

```
▼ {data: {...}, status: 201, statusText: ''}
  ► config: {transitional: {...}, adapter:
  ▼ data:
    body: "테스트 body"
    id: 101
    title: "테스트 Post"
    userId: 10
    ► [[Prototype]]: Object
  ► headers: AxiosHeaders {cache-control:
  ► request: XMLHttpRequest {onreadystate
    status: 201
    statusText: ""
  ► [[Prototype]]: Object
```

# 33. Axios

- postForm (./api/Api.js) – putForm 동일
  - Request Data를 Http Request Parameter에 넣어 전송한다.



```
componentDidMount() {  
  //this.axiosGet();  
  //this.axiosPost();  
  axios.postForm("https://jsonplaceholder.typicode.com/posts", {  
    userId: 10,  
    title: "테스트 Post",  
    body: "테스트 body"  
  })  
  .then(result => {  
    console.log(result);  
  })  
  .catch(error => {  
    console.log(error);  
  });  
}
```

```
▼ {data: {...}, status: 201, statusText: "OK"}  
  ► config: {transitional: {...}, adapter: "xhr", url: "https://jsonplaceholder.typicode.com/posts", data: {userId: 10, title: "테스트 Post", body: "테스트 body"}, headers: {Accept: "application/json", "Content-Type": "application/json"}, method: "post"}  
  ▼ data:  
    id: 101  
    ► [[Prototype]]: Object  
  ► headers: AxiosHeaders {cache-control: "no-cache", "content-type": "application/json", status: 201, statusText: "OK"}  
  ► request: XMLHttpRequest {onreadystatechange: function, readyState: 4, status: 201, statusText: "OK", response: {id: 101, title: "테스트 Post", body: "테스트 body"}, responseText: "[{"id":101,"title":"테스트 Post","body":"테스트 body"}]"}  
  ► [[Prototype]]: Object
```

# 33. Axios

- postForm – Files Posting
  - <https://github.com/axios/axios#files-posting>

## Files Posting

You can easily submit a single file:

```
await axios.postForm('https://httpbin.org/post', {  
  'myVar' : 'foo',  
  'file': document.querySelector('#fileInput').files[0]  
});
```

or multiple files as `multipart/form-data` :

```
await axios.postForm('https://httpbin.org/post', {  
  'files[]': document.querySelector('#fileInput').files  
});
```

`FileList` object can be passed directly:

```
await axios.postForm('https://httpbin.org/post', document.querySelector('#fileInput').files)
```



# 33. Axios

- postForm – HTML Form Posting
  - <https://github.com/axios/axios#-html-form-posting-browser>

## HTML Form Posting (browser)

Pass HTML Form element as a payload to submit it as `multipart/form-data` content.

```
await axios.postForm('https://httpbin.org/post', document.querySelector('#htmlForm'));
```

`FormData` and `HTMLForm` objects can also be posted as `JSON` by explicitly setting the `Content-Type` header to `application/json`:

```
await axios.post('https://httpbin.org/post', document.querySelector('#htmlForm'), {
  headers: {
    'Content-Type': 'application/json'
  }
})
```

# 33. Axios

- validateStatus
  - API 요청의 정상 응답코드를 지정하고 그 외의 경우는 모두 실패로 처리할 수 있도록 한다.

```
componentDidMount() {  
  axios.postForm("https://jsonplaceholder.typicode.com/posts", {  
    userId: 10,  
    title: "테스트 Post",  
    body: "테스트 body"  
  }, {  
    validateStatus: function(status) {  
      console.log("validateStatus > ", status)  
      return status == 200;  
    }  
  })  
  .then(result => {  
    console.log(result);  
  })  
  .catch(error => {  
    console.log(error);  
  });  
}
```

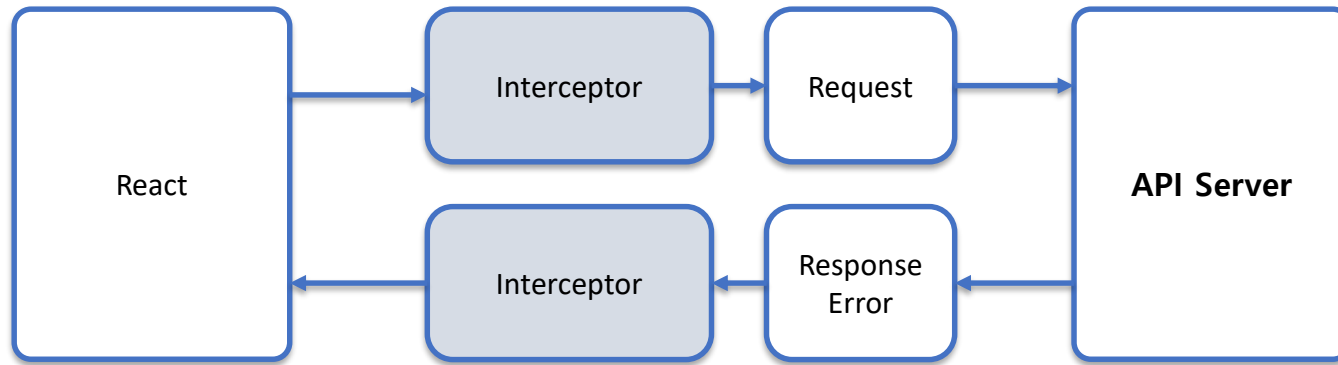
validateStatus > 201

```
▼ AxiosError {message: 'Request failed with status code 201', request, response: {...}, ...} ⓘ  
  ▶ config: {transitional: {...}, adapter: Array(2), transformRequest: [...], transformResponse: [...], timeout: 0, xsrfCookieName: 'XSRF-TOKEN', xsrfHeaderName: 'X-XSRF-TOKEN', ...}  
  ▶ message: "Request failed with status code 201"  
  ▶ name: "AxiosError"  
  ▶ request: XMLHttpRequest {onreadystatechange: null, readyState: 4, status: 201, ...}  
  ▶ response: {data: {...}, status: 201, statusText: 'Created', ...}  
  ▶ stack: "AxiosError: Request failed with status code 201"  
  ▶ [[Prototype]]: Error
```

# 33. Axios

---

- interceptor
  - 요청하기 및 에러응답받기 직전에 코드를 개입시키는 방법



# 33. Axios

- interceptor

```
axios.interceptors.request.use(
  function(config) {
    // 필요한 config 설정을 하거나
    // Spinner 를 띄운다거나 하는 코드를 작성
    return config;
  },
  function(error) {
    // 요청 에러 처리 작성
    return Promise.reject(error);
  }
)

axios.interceptors.response.use(
  function(response) {
    // 정상 응답시 처리할 코드를 작성한다.
    return response;
  },
  function(error) {
    // 응답 에러 처리 작성한다.
    // 예외 응답 등을 받았다면, 응답 관련 Alert 을 띄우는 등
    return Promise.reject(error);
  }
)
```

```
axios.get("https://jsonplaceholder.typicode.com/users")
  .then((result) => {
    console.log(result);
    this.setState( {
      ...this.state,
      isLoading: true,
      items: result.data
    });
  })
  .catch((error) => {
    this.setState({
      ...this.state,
      isLoading: true,
      error
    });
  });
});
```

# React-Router

---

## 34. Router

# 34. Router

# 34. Router

---

- SPA (Single Page Application)
  - 하나의 단일 페이지로 Web App을 구성.
  - Native App과 유사한 UX를 제공할 수 있다.
  - 페이지 내부의 데이터가 변경되면 전체를 Rendering하지 않고 변경된 부분만 갱신하므로 데이터 트래픽을 감소시킬 수 있다.
- MPA (Multi Page Application)
  - 여러 개의 페이지로 Web App을 구성
  - 페이지마다 비슷하거나 동일한 코드가 중첩되어 나타난다.
  - 페이지 내부의 데이터가 변경되면 전체를 갱신해야만 한다.

# 34. Router

---

- Routing?
  - 출발지에서 목적지까지의 경로를 결정하는 기술.
    - 화면에서 화면간의 이동하기 위한 네비게이션 기능
    - 일반적으로 URL 별 페이지를 전환시키는 기능
- React Router 설치
  - `npm install react-router-dom`



# 34. Router

- Routing 맛보기
  - App.js

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Child from "../child/Child";
import MemberList from "../member/MemberList"

function App() {

  return (
    <BrowserRouter>
      <Routes>
        <Route exact={true} path="/" element={<MemberList />} />
        <Route exact={true} path="/members" element={<MemberList />} />
        <Route exact={true} path="/children" element={<Child />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

- exact: url에 작성한 경로와 path의 경로가 정확하게 일치해야만 보이도록 설정한다.

React App x +

← → ↻ ⓘ localhost:3000

이름:

나이:

주소:

React App x +

← → ↻ ⓘ localhost:3000/members

이름:

나이:

주소:

React App x +

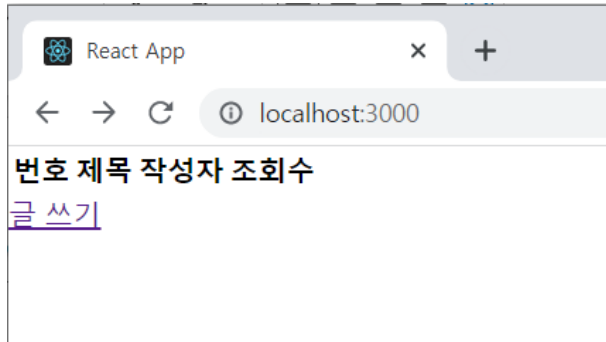
← → ↻ ⓘ localhost:3000/children

3 ▾

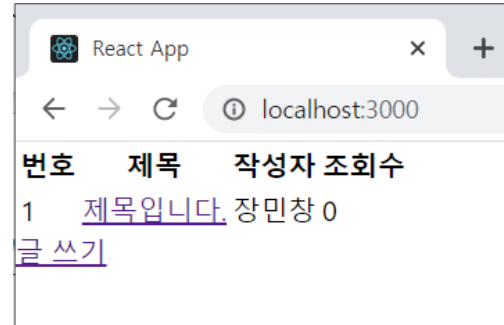
22

# 34. Router

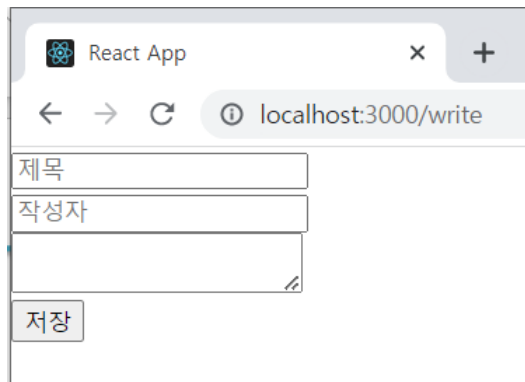
- 게시판 만들기 실습
  - 게시글 (BoardList)



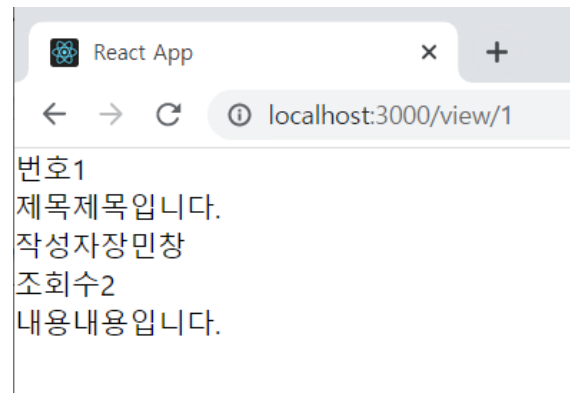
게시글이 있을 때 (BoardList)



- 글쓰기 (Write)



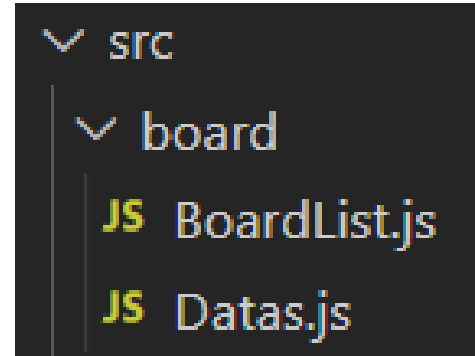
게시글 내용 (Detail)



# 34. Router

---

- 게시판 만들기 실습
  - BoardList.js
    - 게시판의 컴포넌트가 모두 포함된 파일
      - BoardList
      - Row
      - Detail
      - Write
  - Datas.js
    - 임시로 데이터를 저장할 파일
- useNavigate()
- useParams()
- useRef() 사용



# 34. Router

- BoardList

```
export class BoardList extends Component {
  constructor() {
    super();
    this.state = {
      articles: Datas.getAll()
    }
  }

  render() {
    return (
      <>
        <table>
          <thead>
            <tr>
              <th>번호</th>
              <th>제목</th>
              <th>작성자</th>
              <th>조회수</th>
            </tr>
          </thead>
          <tbody>
            {this.state.articles.map((article, idx) => <Row key={idx} detail={article}/>)}
          </tbody>
        </table>
        <div><Link to="/write">글 쓰기</Link></div>
      </>
    );
  }
}
```

# 34. Router

- Row

```
class Row extends Component {  
  render() {  
    const {detail} = this.props;  
  
    return (  
      <tr>  
        <td>{detail.num}</td>  
        <td><Link to={"/view/" + detail.num}>{detail.title}</Link></td>  
        <td>{detail.author}</td>  
        <td>{detail.viewCount}</td>  
      </tr>);  
    }  
  }  
}
```

- Link

- HTML의 Anchor(<A>) 태그의 기능을 수행

# 34. Router

- Write

```
/* useNavigate() 사용을 위해 functional component로 선언 */
export function Write() {
  const nameRef = useRef();
  const titleRef = useRef();
  const descRef = useRef();

  const navigate = useNavigate();

  function save() {
    Datas.push({
      "num": Datas.getAll().length + 1,
      "title": nameRef.current.value,
      "author": titleRef.current.value,
      "desc": descRef.current.value,
      "viewCount": 0
    });
    navigate("/") // "/" Route로 이동
  }

  return (
    <div>
      <div><input type="text" placeholder="제목" ref={nameRef} /></div>
      <div><input type="text" placeholder="작성자" ref={titleRef} /></div>
      <div><textarea ref={descRef} /></div>
      <div><button onClick={save}>저장</button></div>
    </div>
  );
}
```

## useRef()

- ref가 지정된 Element를 직접 제어할 수 있도록 함.

## useNavigate()

- URL을 조작할 수 있도록 함.

# 34. Router

- Detail

```
/* useParams() 사용을 위해 functional component로 선언 */
export function Detail() {
  const {num} = useParams(); // :num 값을 받아온다.
  const detail = Datas.get(num);
  detail.viewCount += 1;
  return (
    <div>
      <div>
        <span>번호</span>
        <span>{detail.num}</span>
      </div>
      <div>
        <span>제목</span>
        <span>{detail.title}</span>
      </div>
      <div>
        <span>작성자</span>
        <span>{detail.author}</span>
      </div>
      <div>
        <span>조회수</span>
        <span>{detail.viewCount}</span>
      </div>
      <div>
        <span>내용</span>
        <span>{detail.desc}</span>
      </div>
    </div>
  );
}
```

## useParams()

- pathVariable 값을 가져올 수 있음

# 34. Router

- App.js

```
import { BrowserRouter, Route, Routes } from "react-router-dom";
import { BoardList, Detail, Write } from "../board/BoardList";

function App() {

  return (
    <BrowserRouter>
      <Routes>
        <Route exact={true} path="/" element={<BoardList />} />
        <Route exact={true} path="/write" element={<Write />} />
        <Route exact={true} path="/view/:num" element={<Detail />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

- exact

- Route Path 값이 정확히 일치하는 Route를 사용하도록 함.



# 34. Router

---

- 조회수가 두 번씩 올라간다?
  - 개발시, React 점검 도구가 실행을 두 번씩 하게 되는 현상
  - index.js 의 <React.StrictMode> 를 삭제한다.

```
const root = ReactDOM.c
root.render(
  <React.StrictMode>
  | <App />
  </React.StrictMode>
);
```

# 34. Router

---

- Path variable 을 query string으로 변경해보기 실습
  - QueryString 설치
    - npm install query-string
  - App.js

```
<Route exact={true} path="/view?" element={<Detail />} />
```

- BoardList.js > Row

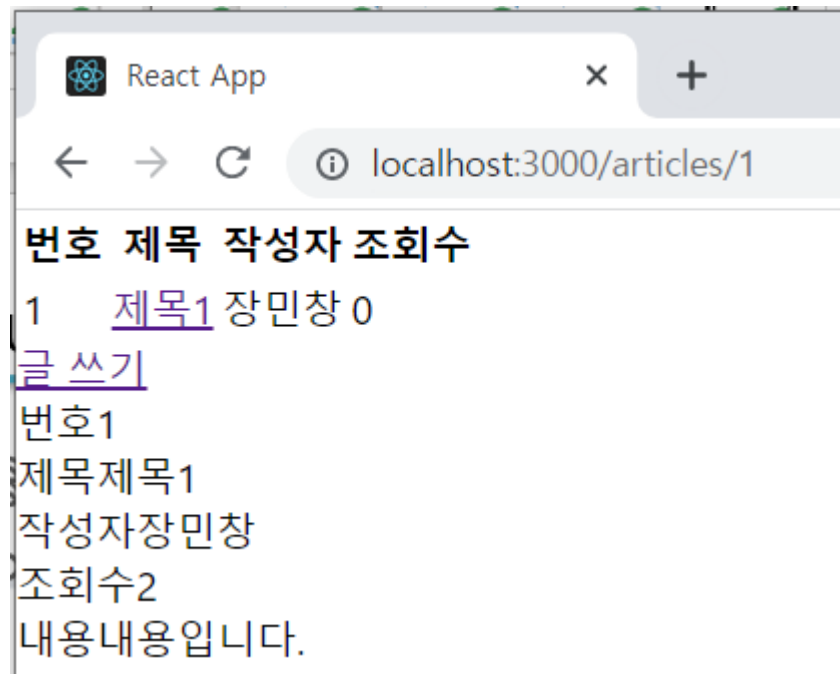
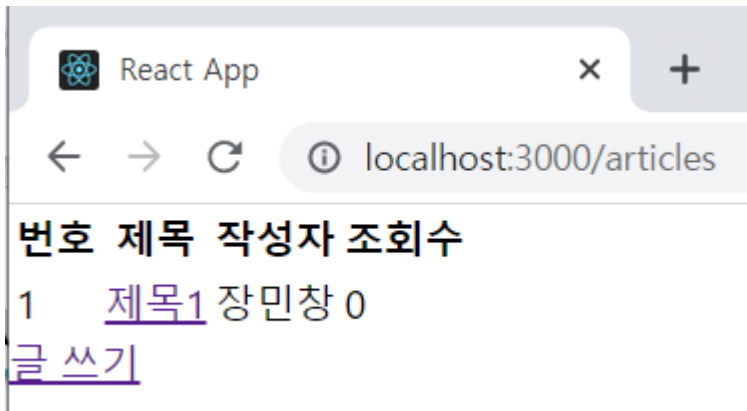
```
<td><Link to={"/view?num=" + detail.num}>{detail.title}</Link></td>
```

- BoardList.js > Detail

```
const {num} = queryString.parse(window.location.search);
```

# 34. Router

- 중첩 라우팅 (Outlet 이용)
  - 여러 레이아웃 동시에 보여주기



# 34. Router

- 중첩 라우팅 (Outlet 이용)
  - App.js

```
function App() {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/articles/*" element={<BoardList />}>  
          <Route path=":num" element={<Detail />} />  
        </Route>  
        <Route exact={true} path="/articles/write" element={<Write />} />  
      </Routes>  
    </BrowserRouter>  
  );  
}
```

# 34. Router

- 중첩 라우팅 (Outlet 이용)
  - BoardList.js > BoardList > render

```
render() {  
  return (  
    <>  
      <table>  
        <thead>  
          <tr>  
            <th>번호</th>  
            <th>제목</th>  
            <th>작성자</th>  
            <th>조회수</th>  
          </tr>  
        </thead>  
        <tbody>  
          {this.state.articles.map((article, idx) => <Row key={idx} detail={article}/>)}  
        </tbody>  
      </table>  
      <div><Link to="/articles/write">글 쓰기</Link></div>  
      <Outlet />  
    </>  
  );  
}
```

# 34. Router

- 중첩 라우팅 (Outlet 이용)
  - BoardList.js > Row > render

```
render() {  
  const {detail} = this.props;  
  
  return (  
    <tr>  
      <td>{detail.num}</td>  
      <td><Link to={"/articles/" + detail.num}>{detail.title}</Link></td>  
      <td>{detail.author}</td>  
      <td>{detail.viewCount}</td>  
    </tr>);  
}
```

# 34. Router

- 중첩 라우팅 (Outlet 이용)
  - BoardList.js > Detail

```
/* useParams() 사용을 위해 functional component로 선언 */  
export function Detail() {  
  const {num} = useParams();  
  const detail = Datas.get(num);  
  detail.viewCount += 1;  
}
```

- BoardList.js > Write > save()

```
function save() {  
  Datas.push({  
    "num": Datas.getAll().length + 1,  
    "title": nameRef.current.value,  
    "author": titleRef.current.value,  
    "desc": descRef.current.value,  
    "viewCount": 0  
  });  
  navigate("/articles") // "/" Route로 이동  
}
```

# 34. Router

- 중첩 라우팅 (Routes 이용)
  - App.js

```
function App() {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/articles/*" element={<BoardList />} />  
        <Route exact={true} path="/articles/write" element={<Write />} />  
      </Routes>  
    </BrowserRouter>  
  );  
}
```



# 34. Router

- 중첩 라우팅 (Routes 이용)
  - BoardList.js > BoardList > render

```
return (

| 번호   | 제목 | 작성자 | 조회수 |
|--|----|-----|-----|
| {this.state.articles.map((article, idx) => <Row key={idx} detail={article}/>)} |    |     |     |



<Link to="/articles/write">글 쓰기</Link></div><Routes><Route path="/:num" element={<Detail />} /></Routes></div>);


```

# Redux

---

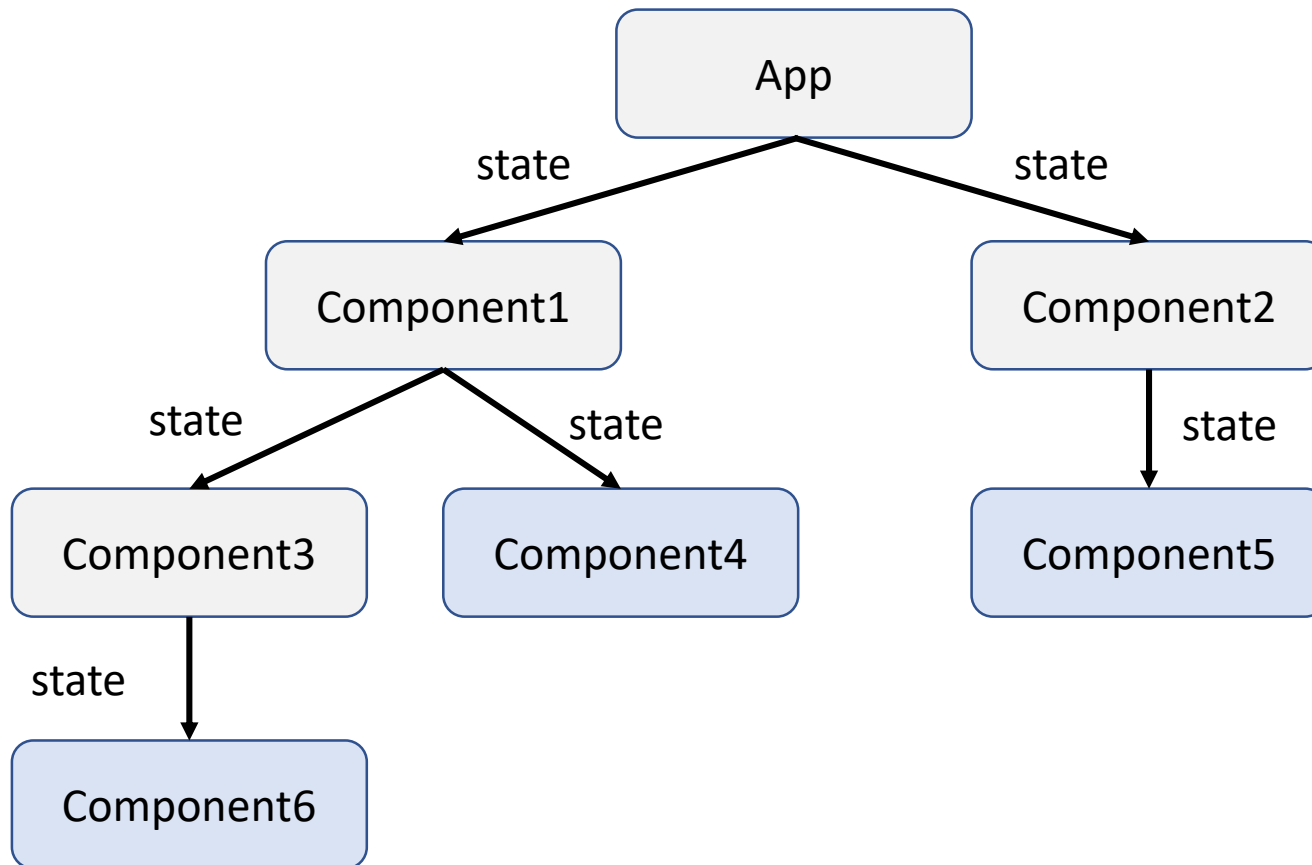
35. Redux

36. Redux Toolkit

# 35. Redux

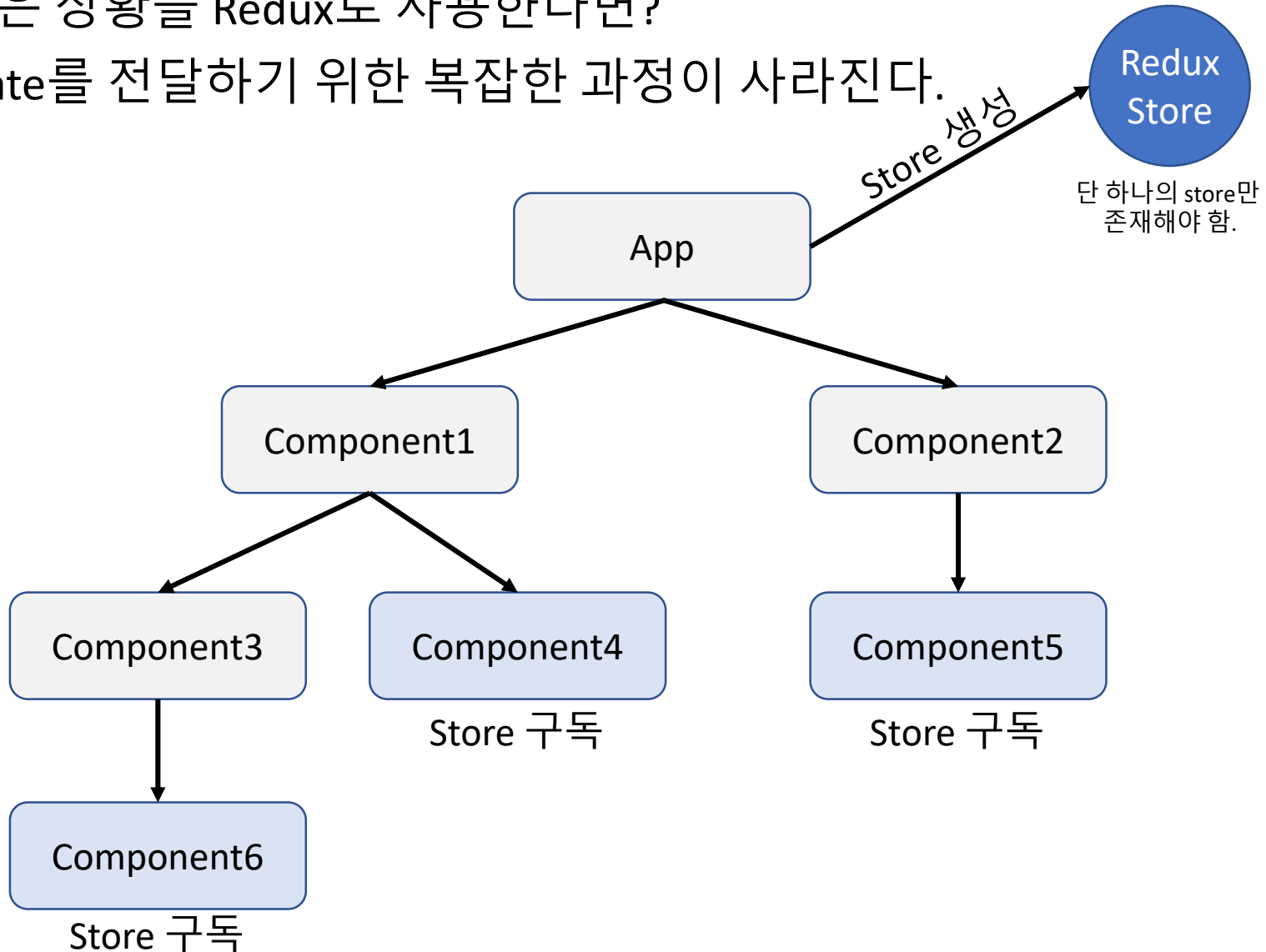
# 35. Redux

- App 에서 Component4, Component5, Component5 로 LoginData State를 전달하려면..
- State를 모두 전달해야만 한다. 만약 컴포넌트가 수십개라면?



# 35. Redux

- 같은 상황을 Redux로 사용한다면?
- State를 전달하기 위한 복잡한 과정이 사라진다.

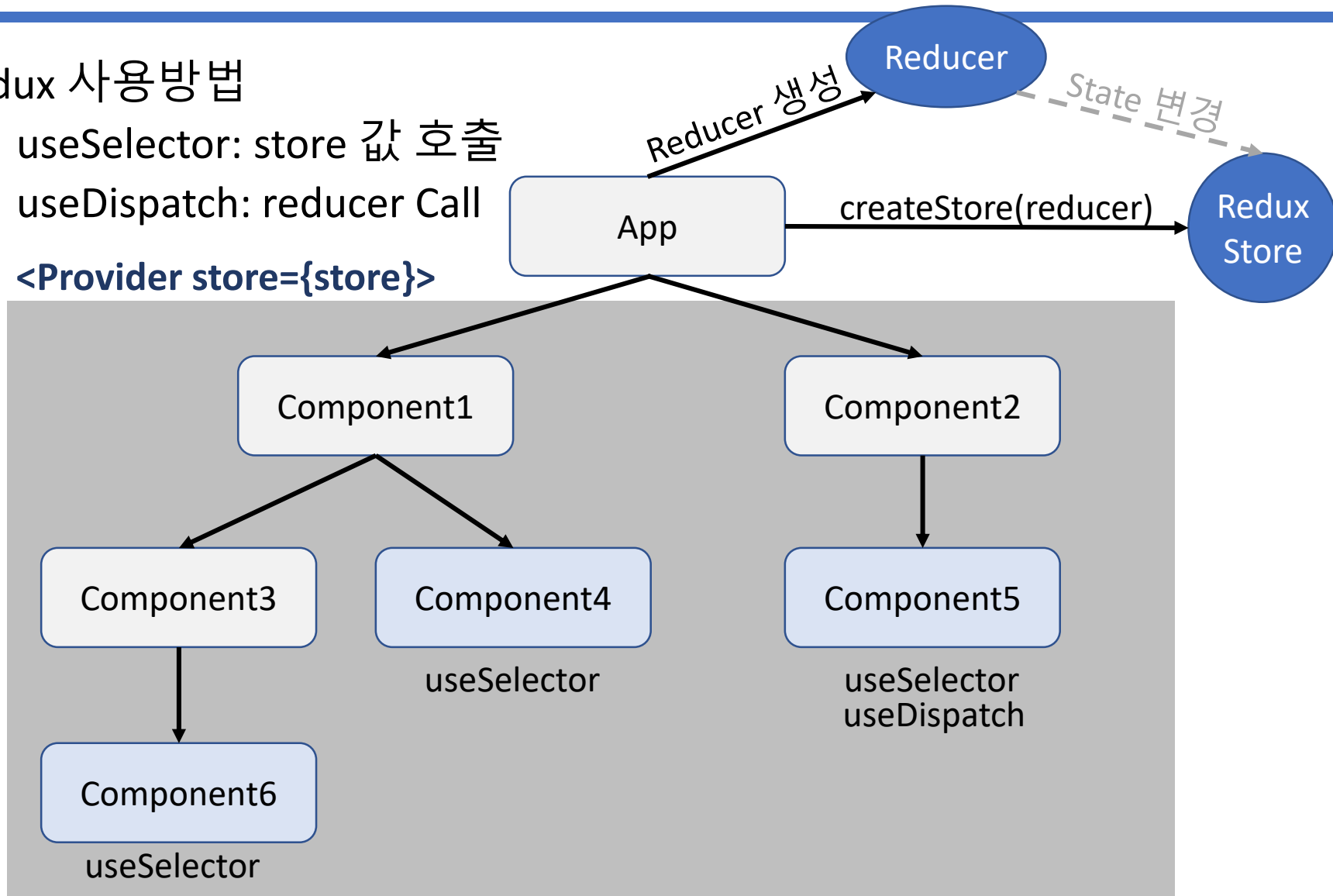


# 35. Redux

- Redux 사용방법

- useSelector: store 값 호출
- useDispatch: reducer Call

**<Provider store={store}>**



**</Provider>**

# 35. Redux

---

- Redux 설치
  - redux와 react-redux 를 설치
  - `npm install redux react-redux`
- redux
  - State를 보관할 Store 관리 모듈
- react-redux
  - React에서 redux를 편하게 사용하기 위해 만든 Adapter

# 35. Redux

- Reducer모듈 생성

Reducer



```
src > redux_modules > JS redux_reducer.js > ...
```

```
1  export default function(currentState = {
2    |   userId: null,
3    |   userName: "비회원"
4  }, action) {
5    |   var newState = {...currentState};
6    |   if (action.type == "LOGIN") {
7    |       |   newState.userId = action.payload.userId;
8    |       |   newState.userName = action.payload.userName;
9    |   }
10   |   else if (action.type == "LOGOUT") {
11   |       |   newState.userId = null;
12   |       |   newState.userName = "비회원";
13   |   }
14   |   return newState;
15  }
```



# 35. Redux

- Store모듈 생성



```
src > redux_modules > JS redux_store.js > ...
1  import { createStore } from "redux"
2  import ReduxReducer from "../redux_reducer"
3
4  export default function ReduxStore() {
5
6      return function() {
7          |       return createStore(ReduxReducer);
8          |
9          |
10     }
```

# 35. Redux

- Store 생성

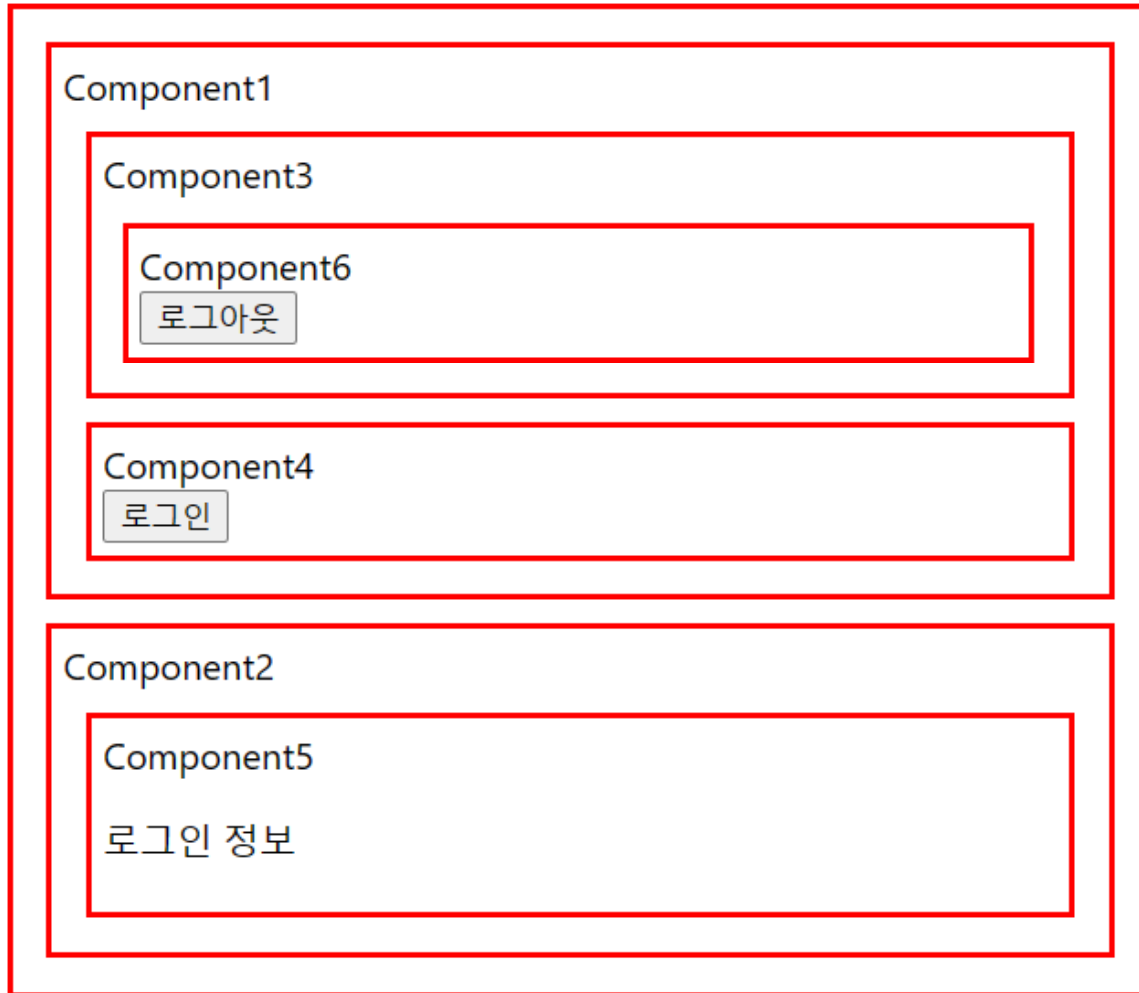


```
src > JS App.js > ...
 1  import { Provider } from "react-redux";
 2  import ReduxStore from "../redux_modules/redux_store";
 3
 4  const store = ReduxStore();
 5
 6  export default function App() {
 7    return (
 8      <>
 9        <Provider store={store}>
10
11      </Provider>
12    </>
13  );
14 }
```

# 35. Redux

---

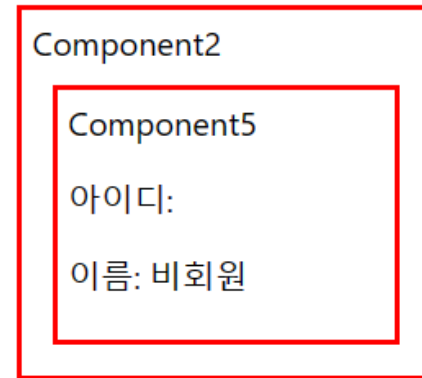
- 각종 하위 컴포넌트 생성
  - ./components/Components.js



# 35. Redux

- Component5 > 로그인 정보에 Store 값 출력해보기

```
41 function Component5() {  
42     const {userId, userName} = useSelector((state) => {return {...state}});  
43     return (  
44         <div>  
45             Component5  
46             <p>아이디: {userId}</p>  
47             <p>이름: {userName}</p>  
48         </div>  
49     );  
50 }
```



- Store의 값을 가져오려면 useSelector를 사용한다.
- useSelector에서 원본 state를 그대로 받아올 수 없다.
  - 복제 state를 가져오거나 state 내부의 필드만 가져올 수 있다.
- useSelector는 state를 인자로 가지는 함수를 작성해야 한다.

# 35. Redux

- Component4 > 로그인 클릭하면 로그인 정보 변경해보기

```
31 function Component4() {  
32     const dispatch = useDispatch();  
33     const loginAction = {type: "LOGIN", payload: {userId: "member1", userName: "홍길동"}};  
34     return (  
35         <div>  
36             Component4  
37             <br/>  
38             <button onClick={() => dispatch(loginAction)}>로그인</button>  
39         </div>  
40     );  
41 }
```

- useDispatch() 는 Reducer에 Action을 전달해 Store 데이터를 변경한다.

# 35. Redux

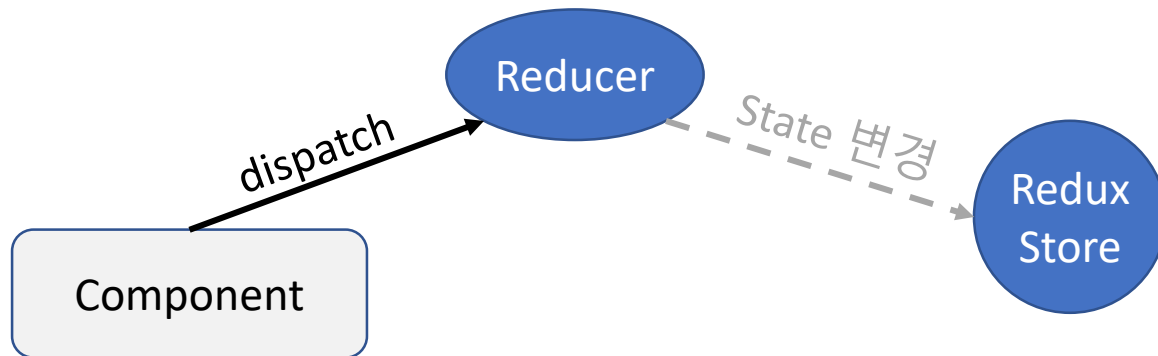
- Component6 > 로그아웃 클릭하면 로그인 정보 변경해보기

```
54  function Component6() {  
55      const dispatch = useDispatch();  
56      const logoutAction = {type: "LOGOUT", payload: {}};  
57      return (  
58          <div>  
59              Component6  
60              <br/>  
61              <button onClick={() => dispatch(logoutAction)}>로그아웃</button>  
62          </div>  
63      );  
64  }
```

- useDispatch() 는 Reducer에 Action을 전달해 Store 데이터를 변경한다.

# 35. Redux

- useDispatch() 는 Reducer에 Action을 전달해 Store 데이터를 변경한다.
- 변경 후 Store를 구독하는 Component만 Re-Rendering 된다.



# 36. Redux Toolkit



# 36. Redux Toolkit

---

- redux
  - state 관리 모듈
  - react에서 직접 사용하기가 어려움
- react-redux
  - react에서 redux를 편하게 사용하기 위한 모듈
  - redux를 직접 사용하는 것보다 편하다.
- redux-toolkit
  - react-redux 보다 더 편하게 사용하기 위한 모듈
  - react-redux 의 확장 버전
    - (여러 개의 store를 사용할 수 있다)
      - Store (Slice Store1, Slice Store2, ...)

# 36. Redux Toolkit

---

- redux-toolkit 모듈 설치
  - `npm install @reduxjs/toolkit`

# 36. Redux Toolkit

- redux-toolkit 특징 1



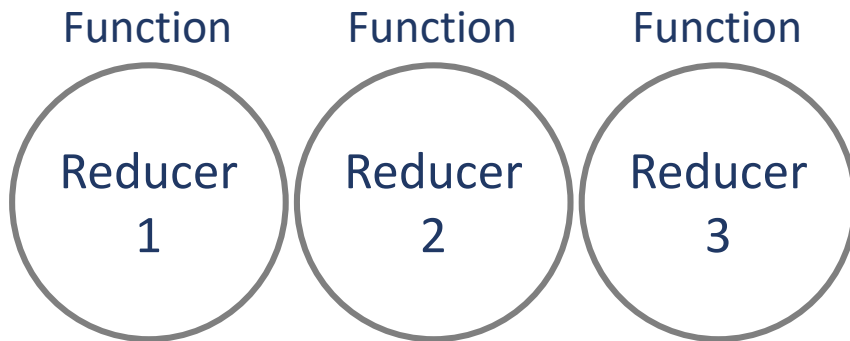
Redux  
React-Redux



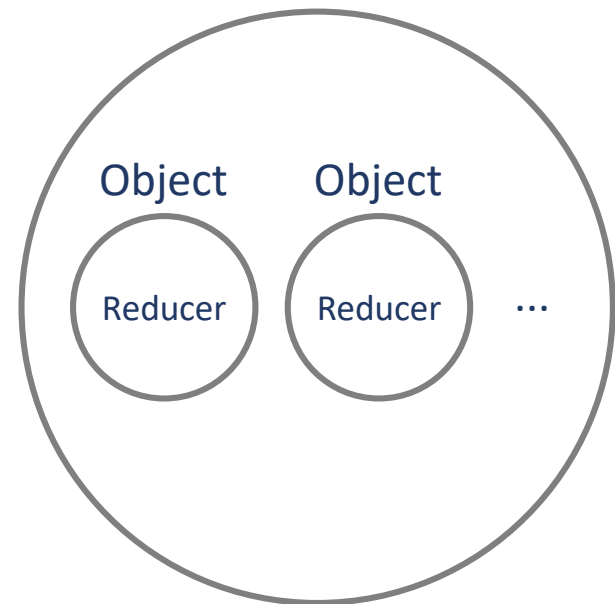
Redux-Toolkit

# 36. Redux Toolkit

- redux-toolkit 특징 2



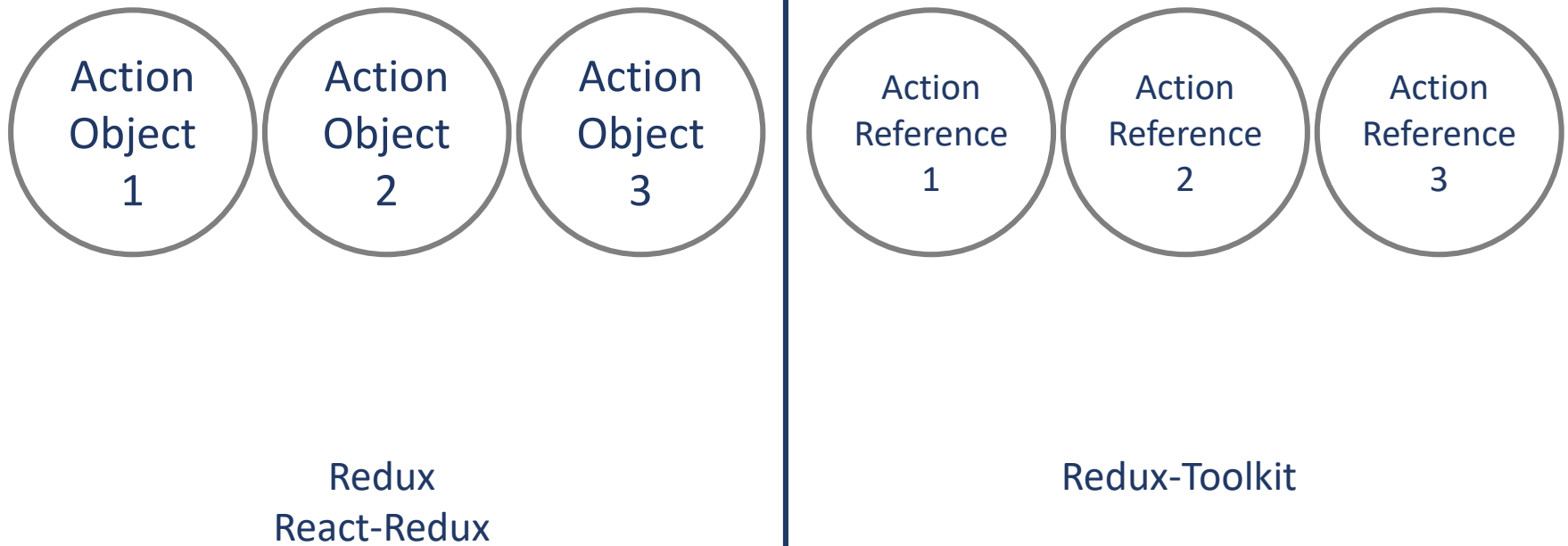
Redux  
React-Redux



Redux-Toolkit

# 36. Redux Toolkit

- redux-toolkit 특징 3



# 36. Redux Toolkit

- Slice Store 생성

```
src > redux_toolkit_modules > JS UserSliceStore.js > ...  
1  import { createSlice } from "@reduxjs/toolkit";  
2  
3  export const userSlice = createSlice({  
4    name: "userSlice",  
5    initialState: {userId: null, userName: "비회원"},  
6    reducers: {  
7      login: function(state, action) {  
8        state.userId = action.payload.userId;  
9        state.userName = action.payload.userName;  
10     },  
11     logout: function(state, action) {  
12       state.userId = null;  
13       state.userName = "비회원";  
14     }  
15   }  
16 });
```

- name: 슬라이스 스토어 명
- initialState: 스토어에 저장될 state의 기본 값
- reducers: userSlice의 reducer 정의

# 36. Redux Toolkit

- Store 생성

```
src > redux_toolkit_modules > JS ReduxStore.js > ...  
1  import { configureStore } from "@reduxjs/toolkit";  
2  import { userSlice } from "../UserSliceStore";  
3  
4  export const store = configureStore({  
5    |    reducer: {  
6    |      |    user: userSlice.reducer  
7    |      |  
8    |    }  
    });
```

- reducer: Store에 저장할 SliceStore의 모음
- reducer.user: state에 저장할 UserSlice 의 키
  - state.user 로 접근이 가능하다.
- userSlice.reducer: userSlice에 정의된 reducers를 하나로 모은 객체

# 36. Redux Toolkit

- Provider에 store 지정

src > JS App.js > ...

```
1  import { Provider } from "react-redux";
2  import { Component1, Component2 } from "../components/Components";
3
4  import "../App.css";
5  import { store } from "../redux_toolkit_modules/ReduxStore";
6
7  export default function App() {
8    return (
9      <>
10     <Provider store={store}>
11       <Component1 />
12       <Component2 />
13     </Provider>
14   </>
15 );
16 }
```



## 36. Redux Toolkit

- Store 값 출력

```
function Component5() {  
  // const {userId, userName} = useSelector((state) => {return {...state}});  
  const {userId, userName} = useSelector((state) => {return {...state.user}});  
  console.log(userId, userName);  
  return (  
    <div>  
      Component5  
      <p>아이디: {userId}</p>  
      <p>이름: {userName}</p>  
    </div>  
  );  
}
```

## 36. Redux Toolkit

- 로그인 처리

```
function Component4() {
  const dispatch = useDispatch();
  // const loginAction = {type: "LOGIN", payload: {userId: "member1", userName: "홍길동"}};
  const payload = {userId: "member1", userName: "홍길동"};
  return (
    <div>
      Component4
      <br/>
      <button onClick={() => dispatch(userSlice.actions.login(payload))}>로그인</button>
    </div>
  );
}
```

## 36. Redux Toolkit

- 로그아웃 처리

```
function Component6() {  
  const dispatch = useDispatch();  
  // const logoutAction = {type: "LOGOUT", payload: {}};  
  return (  
    <div>  
      Component6  
      <br/>  
      <button onClick={() => dispatch(userSlice.actions.logout())}>로그아웃</button>  
    </div>  
  );  
}
```

# **37. Mini Project**

# 37. Mini Project

- 영화 목록 Application
  - 장르별 영화목록

ktds FLIX

영화 검색

액션



모험

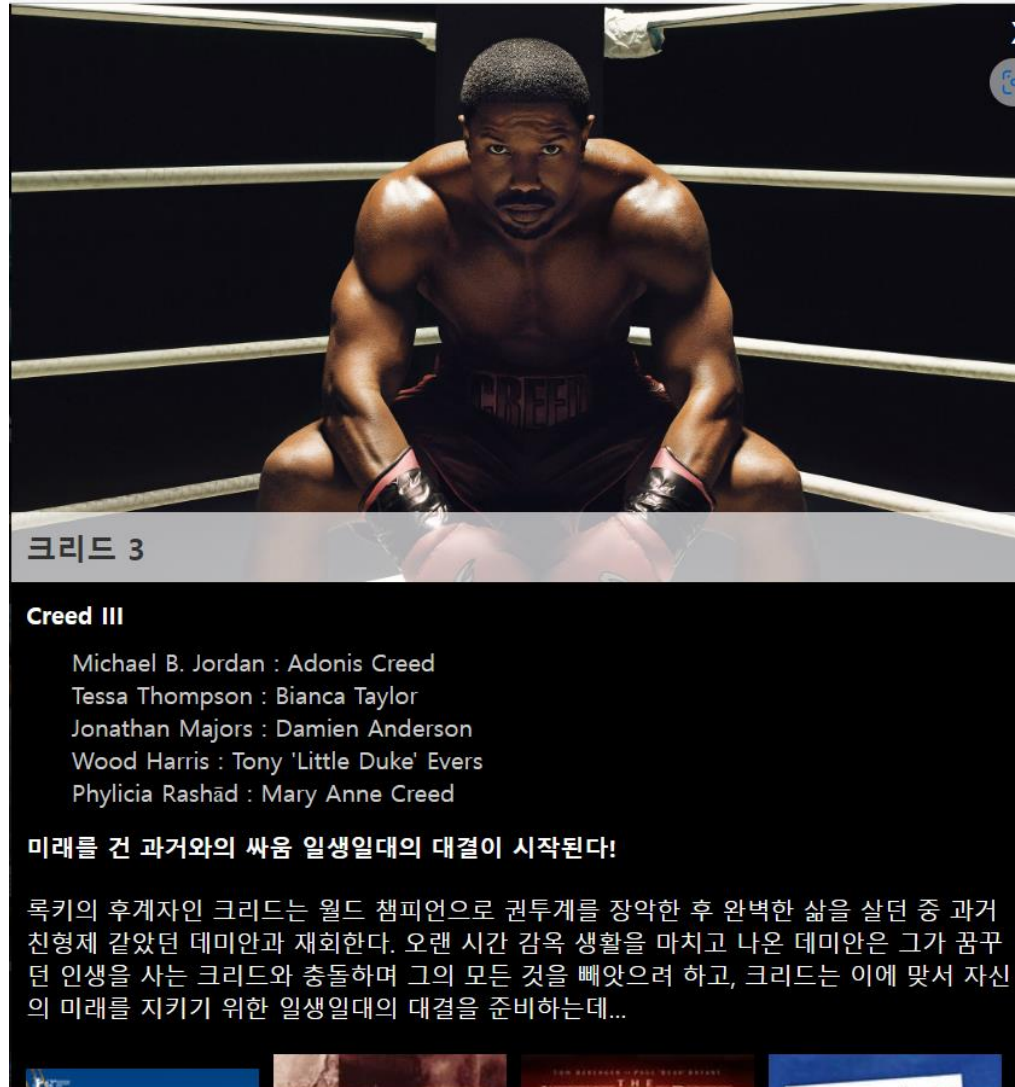


애니메이션



# 37. Mini Project

- 영화 목록 Application
  - 영화상세보기



# 37. Mini Project

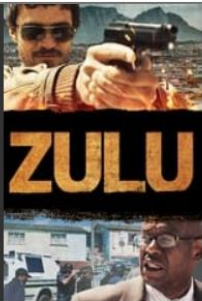
- 영화 목록 Application
  - 영화 검색

ktids FLIX

영화 검색

검색

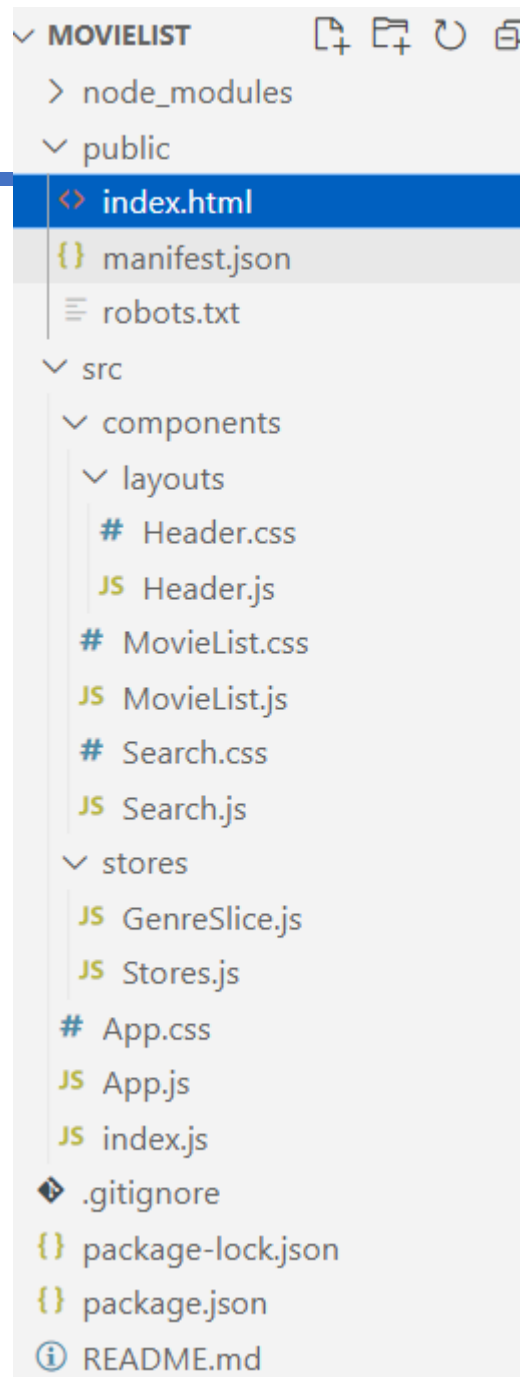
범죄도시





# 37. Mini Project

- 영화 목록 Application
  - 프로젝트 구조





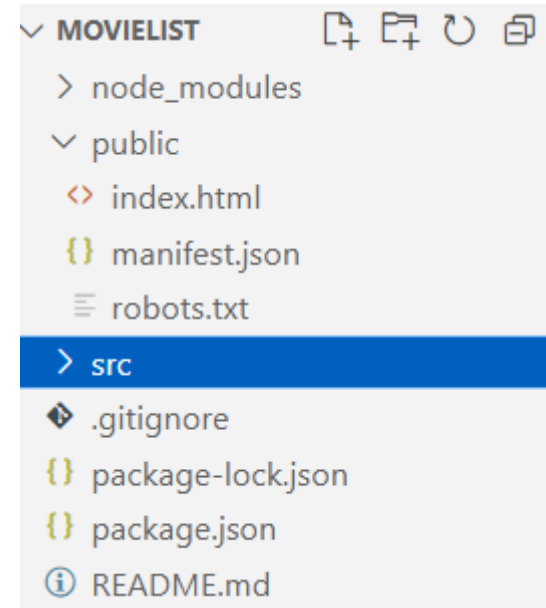
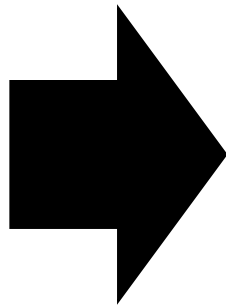
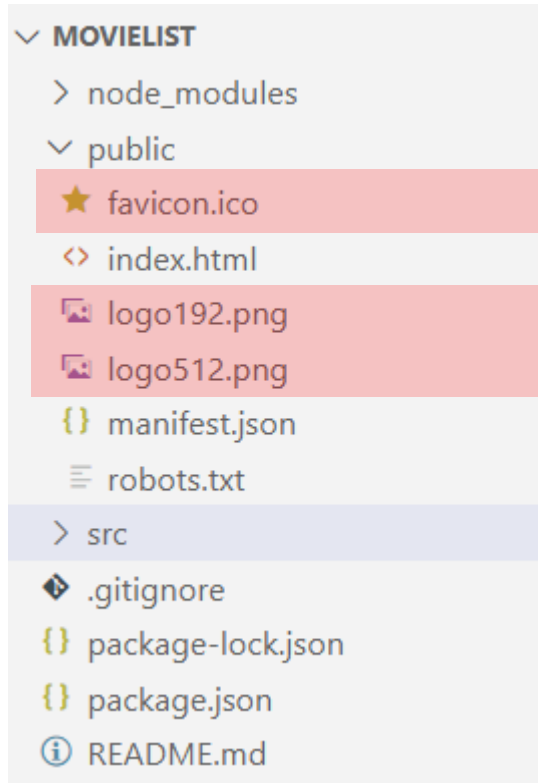
# 37. Mini Project

---

- 준비작업
  - Project 명
    - movielist
  - [TMDB] <https://www.themoviedb.org/?language=ko>
    - 회원가입
    - 이메일 인증
    - API 키 발급
    - API 확인
  - 필요 모듈 설치
    - axios
    - react-router, react-router-dom
    - redux
    - react-redux
    - @reduxjs/toolkit (redux-toolkit)

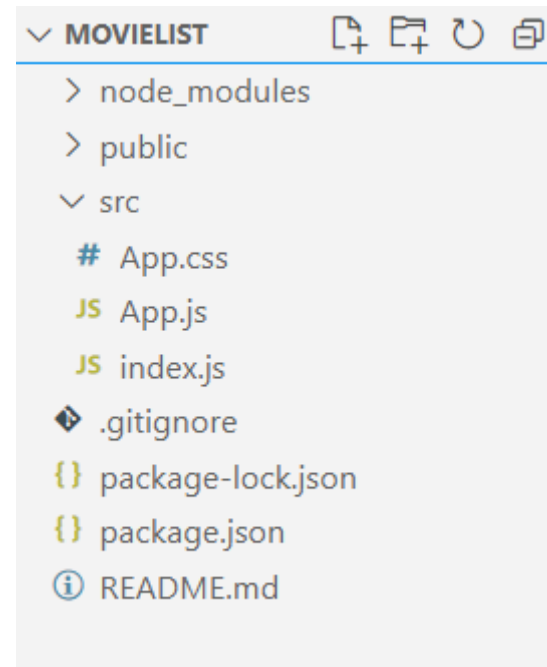
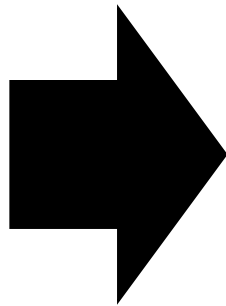
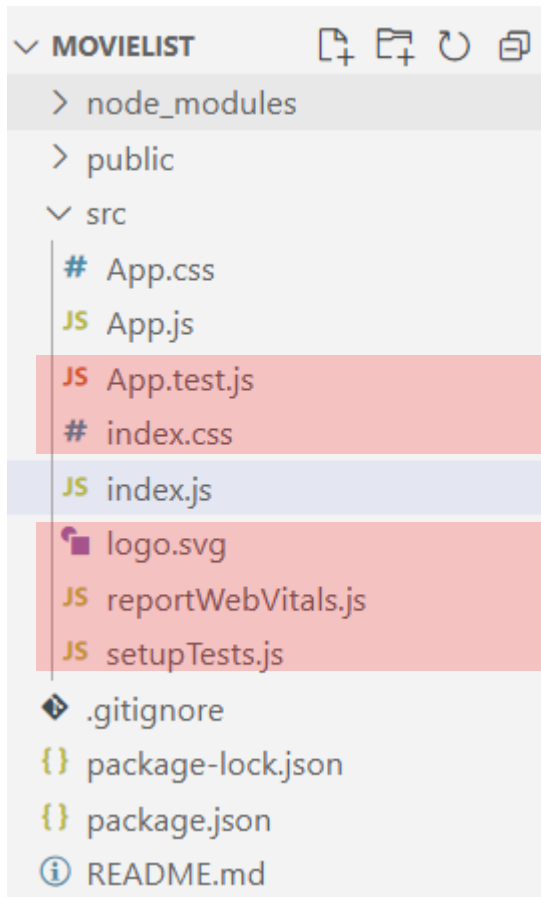
# 37. Mini Project

- 준비작업
  - 필요 없는 파일 제거



# 37. Mini Project

- 준비작업
  - 필요 없는 파일 제거



# 37. Mini Project

- 준비작업
  - index.js 수정

```
JS index.js ×
src > JS index.js > ...
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import './index.css';
4  import App from './App';
5  import reportWebVitals from './reportWebVitals';
6
7  const root = ReactDOM.createRoot(document.getElementById('root'));
8  root.render(
9    <React.StrictMode>
10     <App />
11   </React.StrictMode>
12 );
13
14 // If you want to start measuring performance in your app, pass a function
15 // to log results (for example: reportWebVitals(console.log))
16 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
17 reportWebVitals();
```

# 37. Mini Project

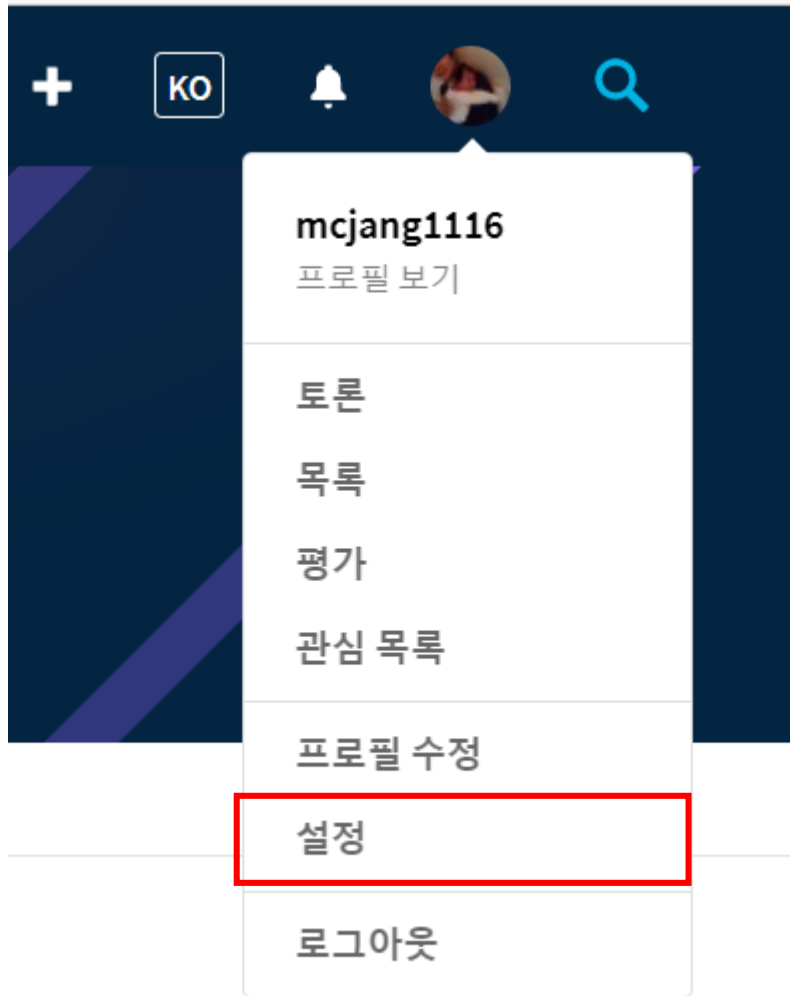
- 준비작업
  - App.js, App.css 수정

```
JS App.js  ×
src > JS App.js > ...
1  import './App.css';
2
3  function App() {
4  ✓  return (
5      <div>
6      |   Hello
7      | </div>
8      );
9  }
10
11  export default App;
```

```
# App.css  ×
src > # App.css
1  |
```

# 37. Mini Project

- TMDB 회원 가입 및 API 키 발급



장르

# 37. Mini Project

- TMDB 회원 가입 및 API 키 발급

## 설정

프로필 편집

계정 설정

Streaming Services

알람 설정

차단 유저

Import List

공유 설정

Sessions

API

계정 삭제

## API Overview 생성

TMDB offers a powerful API service that is fi  
logos for attribution [here](#).

## 문서

Our primary documentation is located at [d](#)

## 지원

If you have questions or comments about t

## API 키 요청

To generate a new API key, [click here](#).

# 37. Mini Project

---

- TMDB 회원 가입 및 API 키 발급



## Developer

- You are an individual
- Your project is still in development
- Your project is non profit
- Your project is ad supported



## Professional

- You represent a company
- Your project is for profit (not ad supported)
- You are an OEM or hardware vendor



# 37. Mini Project

---

- TMDB 회원 가입 및 API 키 발급

이용 형태

웹사이트 ▼

어플리케이션 이름

React Movie App

어플리케이션 URL

http://127.0.0.1:3000

어플리케이션 개요

영화 애플리케이션

## 37. Mini Project

- TMDB 회원 가입 및 API 키 발급

## API 7 (v3 auth)

3cc0c46b7fd6181b12a5d89d2a40c0a8

## API 요청 예

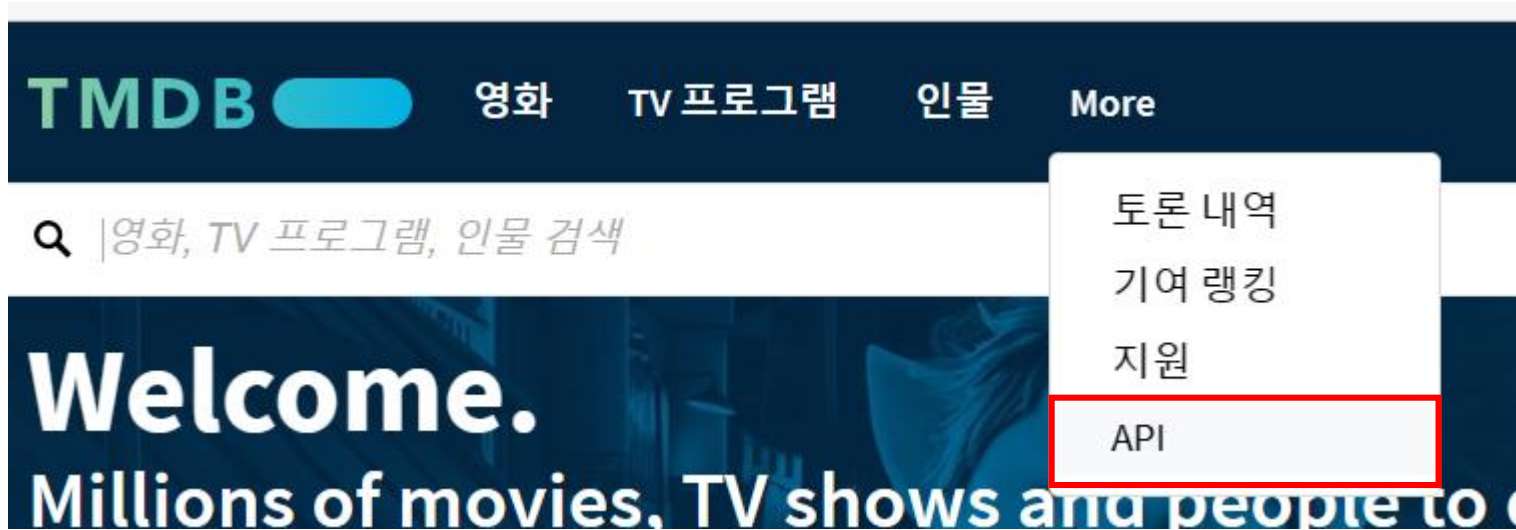
[https://api.themoviedb.org/3/movie/550?api\\_key=3cc0c46b7fd6181b12a5d89d2a40c0a8](https://api.themoviedb.org/3/movie/550?api_key=3cc0c46b7fd6181b12a5d89d2a40c0a8)

## API 읽기 액세스 토큰 (v4 auth)

eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiIyZmVzQ2YjdmZDYxODFiMTJhNWQ4OWQyYTQwYzBhOCIsInN1Yil6IjY0MDI4MmY3Njk5Zml3MDBlNmZmMDE2MnNjb3BlcyI6WyJhcGlcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.Raolgu\_xB4ICulZi4IVLIgBKm5dP8BbPocZJR7Jc1BQ

# 37. Mini Project

- API요청 URL 확인



# 37. Mini Project

---

- API요청 URL 확인

## API Overview

Our API is available for everyone to use. A TMDB user account is required to request an API key  
As always, you must attribute TMDB as the source of your data. Please be sure to read more ab

## API Documentation

---

To view all the methods available, you should head over to [developers.themoviedb.org](https://developers.themoviedb.org). Everyt  
help you understand what is available.

# 37. Mini Project

- 설정 요청주소 확인

## CONFIGURATION

GET [Get API Configuration](#)

GET [Get Countries](#)

GET [Get Jobs](#)

Definition

Try it out

## Variables

api_key	Your TMDb API key	optional
---------	-------------------	----------

## Query String

api_key	<<api_key>>	required
---------	-------------	----------

SEND REQUEST

[https://api.themoviedb.org/3/configuration?api\\_key=<<api\\_key>>](https://api.themoviedb.org/3/configuration?api_key=<<api_key>>)

# 37. Mini Project

- 장르 확인

## GENRES

GET [Get Movie List](#)

Definition Try it out

## Variables

api_key	Your TMDb API key	optional
---------	-------------------	----------

## Query String

api_key	String	required
language	ko-KR	optional

SEND REQUEST

<https://api.themoviedb.org/3/genre/movie/list?language=ko-KR>

# 37. Mini Project

- 영화 목록 확인

DISCOVER

GET Movie Discover

without_genres	string	optional
with_keywords	String	optional
without_keywords	String	optional
with_runtime.gte	value	optional
with_runtime.lte	value	optional
with_original_language	String	optional
with_watch_providers	String	optional
watch_region	String	optional
with_watch_monetization_types	flatrate	optional
without_companies	String	optional

SEND REQUEST

[https://api.themoviedb.org/3/discover/movie?api\\_key=3cc0c46b7fd6181b12a5d89d2a40c0a8&language=ko-KR&region=KR&sort\\_by=popularity.desc&include\\_adult=false&include\\_video=false&page=1&with\\_watch\\_monetization\\_types=flatrate](https://api.themoviedb.org/3/discover/movie?api_key=3cc0c46b7fd6181b12a5d89d2a40c0a8&language=ko-KR&region=KR&sort_by=popularity.desc&include_adult=false&include_video=false&page=1&with_watch_monetization_types=flatrate)

# 37. Mini Project

- 영화 검색 확인

## SEARCH

GET Search Companies

GET Search Collections

GET Search Keywords

GET **Search Movies**

GET Multi Search

GET Search People

GET Search TV Shows

## Variables

api_key	3cc0c46b7fd6181b12a5d89d2a40c0a8	optional
---------	----------------------------------	----------

## Query String

api_key	3cc0c46b7fd6181b12a5d89d2a40c0a8	required
language	en-US	optional
query	String	required
page	1	optional
include_adult	false	optional
region	String	optional
year	Integer	optional
primary_release_year	Integer	optional

SEND REQUEST

[https://api.themoviedb.org/3/search/movie?api\\_key=3cc0c46b7fd6181b12a5d89d2a40c0a8&language=en-US&page=1&include\\_adult=false](https://api.themoviedb.org/3/search/movie?api_key=3cc0c46b7fd6181b12a5d89d2a40c0a8&language=en-US&page=1&include_adult=false)



# 37. Mini Project

---

- 영화 크레딧 확인

## MOVIES

GET Get Details

GET Get Account States

GET Get Alternative Titles

GET Get Changes

GET **Get Credits**

GET Get External IDs

# 감사합니다.

---

ES6로 개발하는 ReactJS

장민창

mcjang@huccloud.co.kr