

React 애플리케이션 개발

최신 React.js로 구현하는
프론트엔드 애플리케이션 개발

장민창

mcjang1116@gmail.com

React 개요

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

1. 리액트 프레임워크

1. 리액트 프레임워크

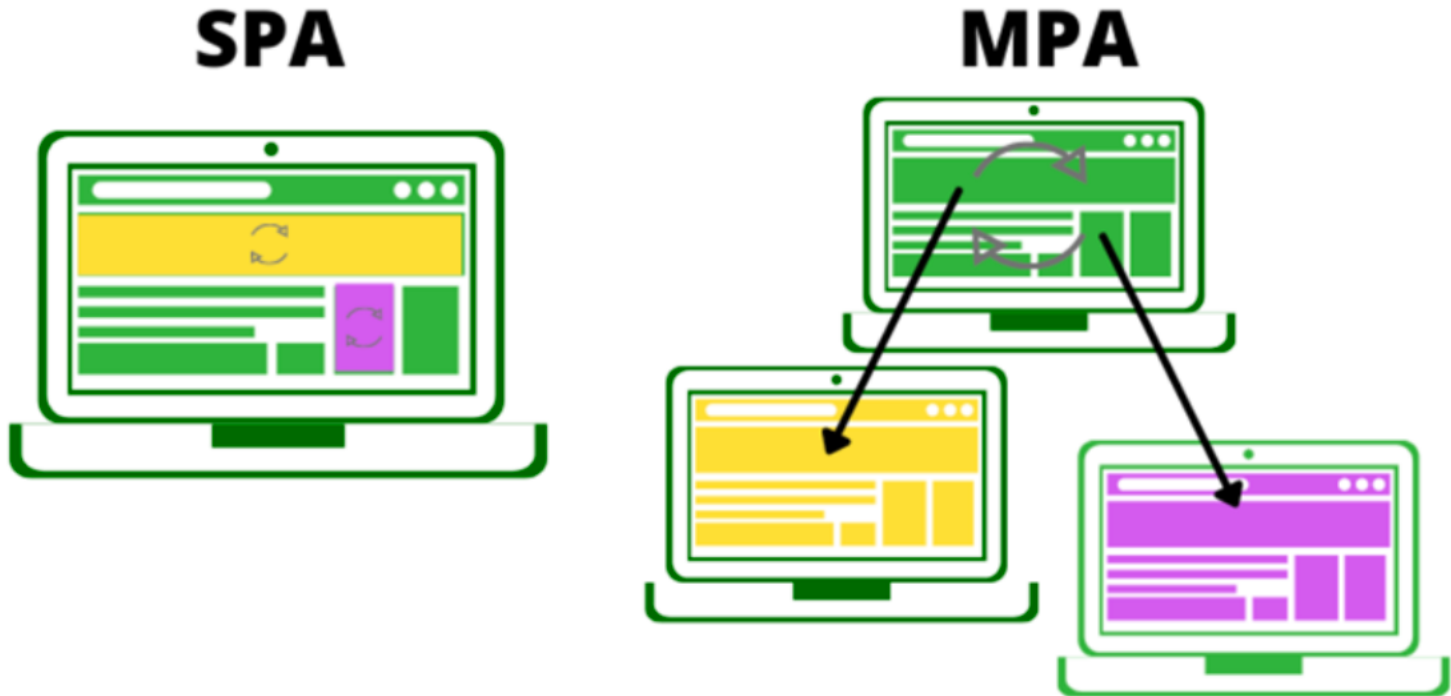
- 리액트 (React)
 - Open-source javascript framework
 - 2013년 페이스북(현재 메타)에서 개발/발표
- SPA 개발 프레임워크
 - SPA (Single Page Application) 기반의 프레임워크
- Javascript 기반의 프레임워크
 - ECMA Script 2015 (ES6) 이상의 문법만을 지원함.

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



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
 - 페이지를 갱신 할 때마다 불필요한 Data-Load가 발생.
- SPA (Single Page Application)
 - WebApp이 하나의 페이지로만 구성된 것.
 - 화면의 요청이 있을 때 마다(URL) 바뀌는 컴포넌트만 갱신
 - 데이터가 변경된 컴포넌트만 갱신하기에 필요한 만큼의 Data-Load를 수행.
 - 단, 렌더링을 클라이언트에서 수행하기에 데이터와 컴포넌트가 매우 많을 경우, 초기 렌더링의 시간이 많이 소요됨.

2. 개발환경 세팅

2. 개발환경 세팅

- node.js 설치 (<https://nodejs.org/ko>)

어디서든 JavaScript를 실행하세요

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

Node.js 다운로드 (LTS) 

Node.js 다운로드 **v22.14.0**¹ LTS. Node.js는 패키지 관리자를 통해서도 다운로드 할 수 있습니다.

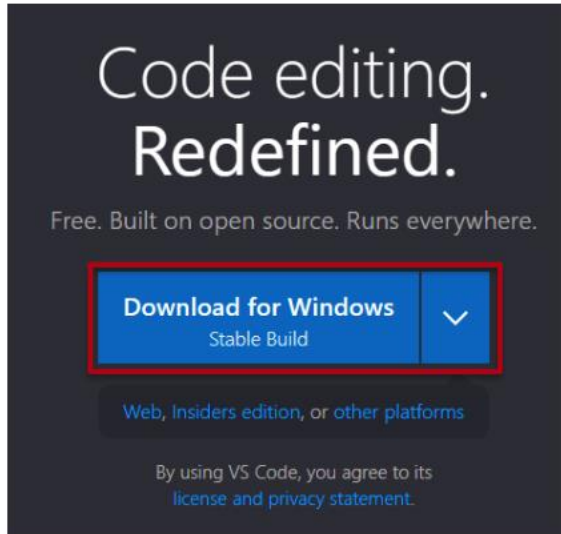
새로운 기능을 먼저 경험하고 싶다면 **Node.js v23.11.0** ¹를 다운로드 받으세요.

Javascript 기반의 서버 프로그램

nodejs 기반의 npm Repository를 활용하기 위해 설치
(React는 npm에 등록된 라이브러리)

2. 개발환경 세팅

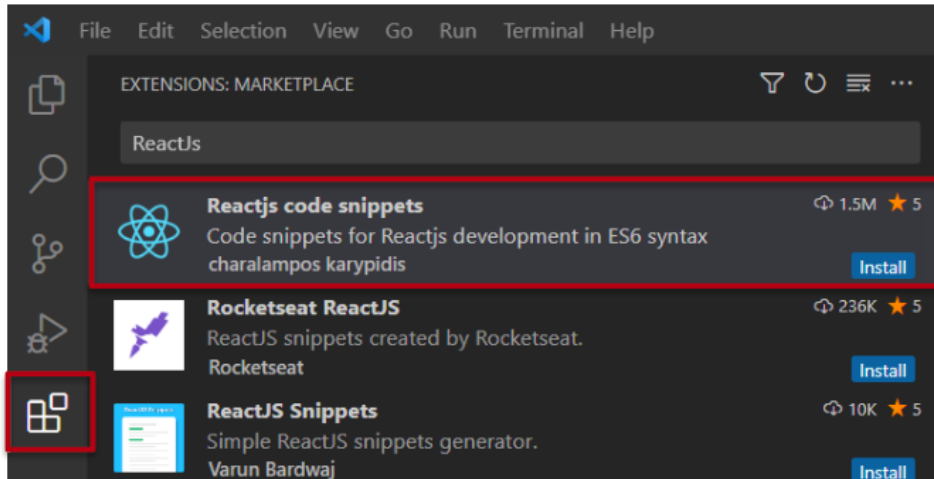
- VSCode 설치 (<https://code.visualstudio.com/>)



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

2. 개발환경 세팅

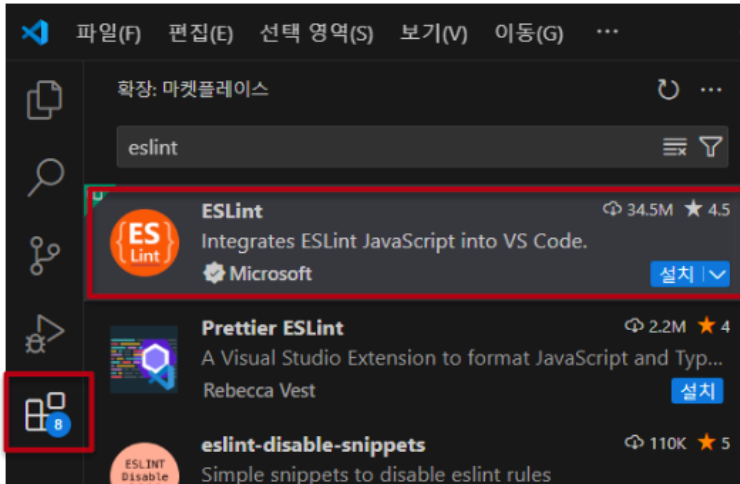
- Reactjs code snippets 플러그인 설치



- Reactjs Content Assist를 지원.

2. 개발환경 세팅

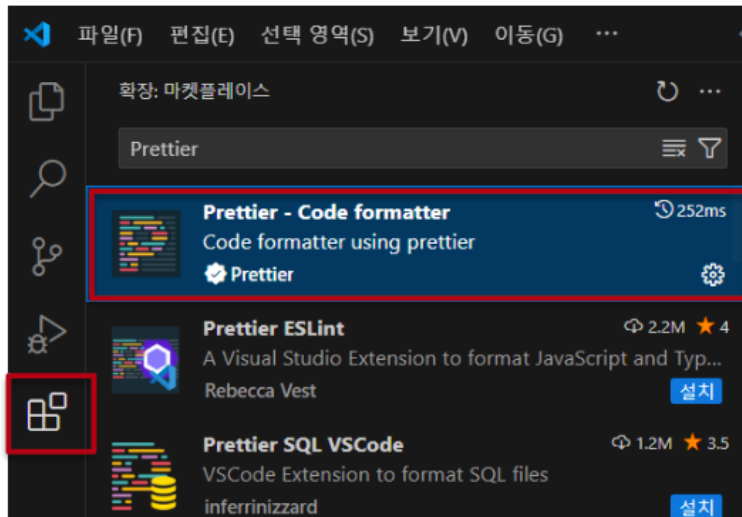
- ESLint 플러그인 설치



- Javascript 문법 오류를 검출함.

2. 개발환경 세팅

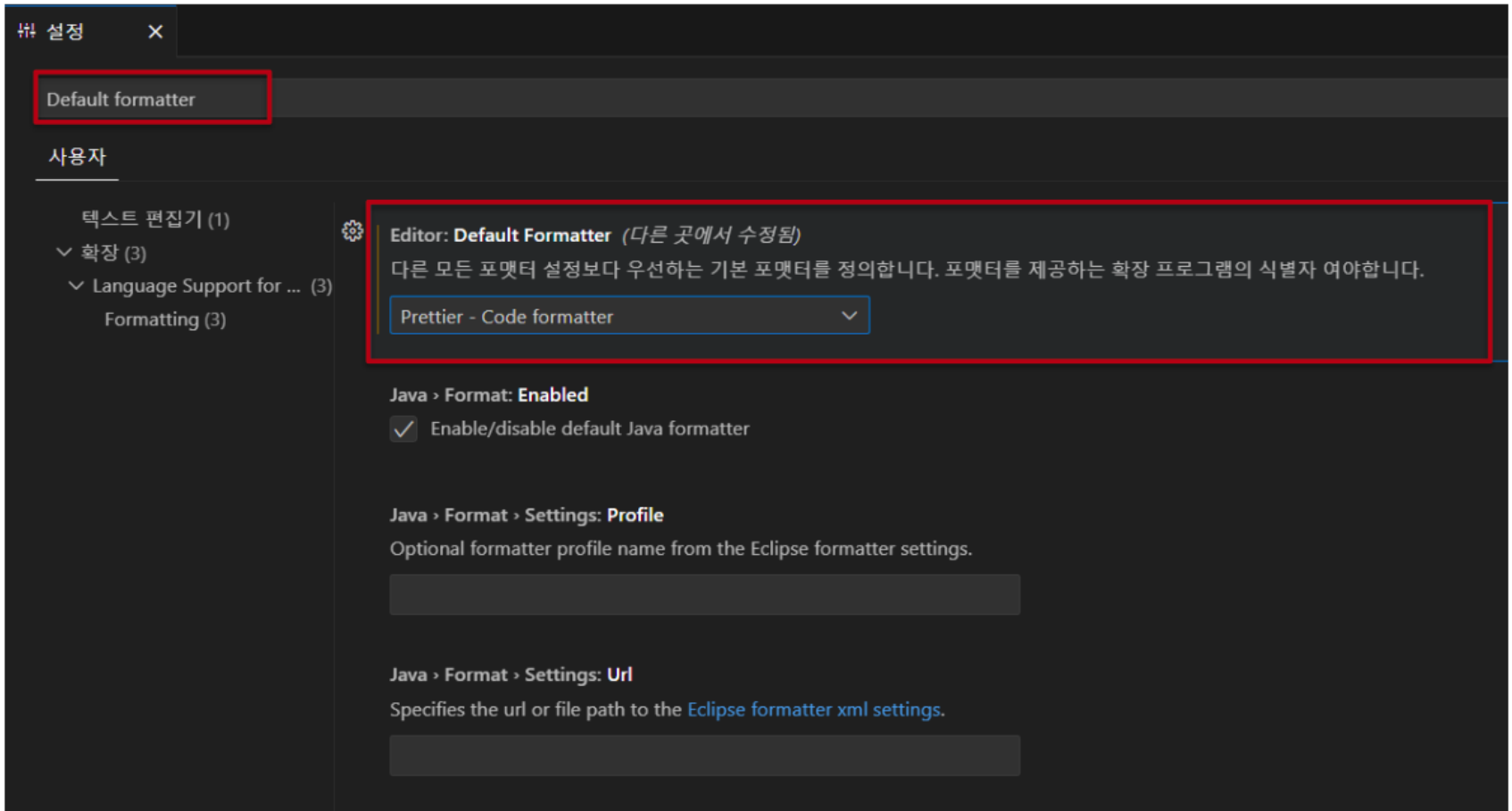
- Prettier 플러그인 설치



- 코드를 자동으로 정렬함.

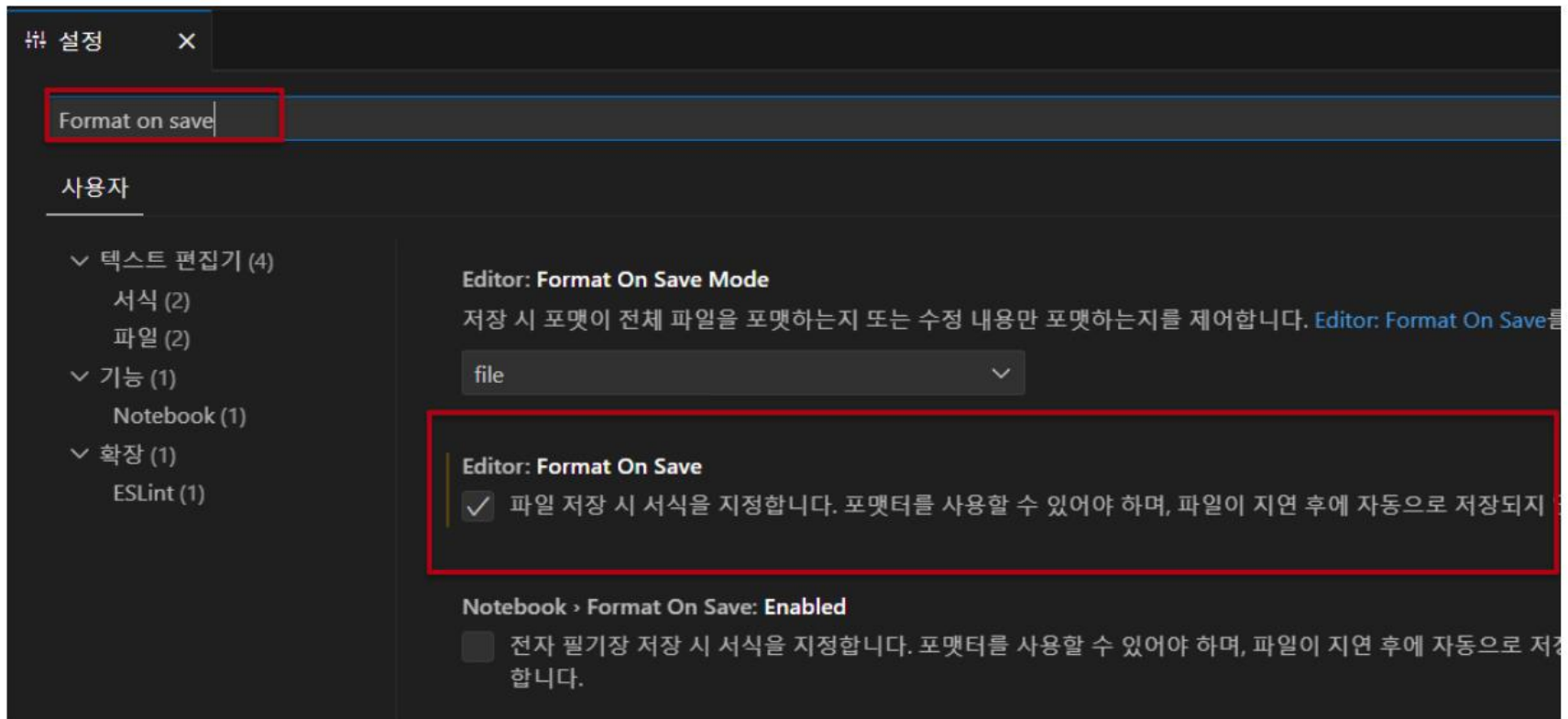
2. 개발환경 세팅

- Prettier 플러그인 활성화 설정
 - 설정에서 Default Formatter 검색
 - Default Foramtter 를 Prettier로 설정



2. 개발환경 세팅

- Prettier 플러그인 활성화 설정
 - 설정에서 Format on save 검색
 - Editor: Format On Save 체크



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

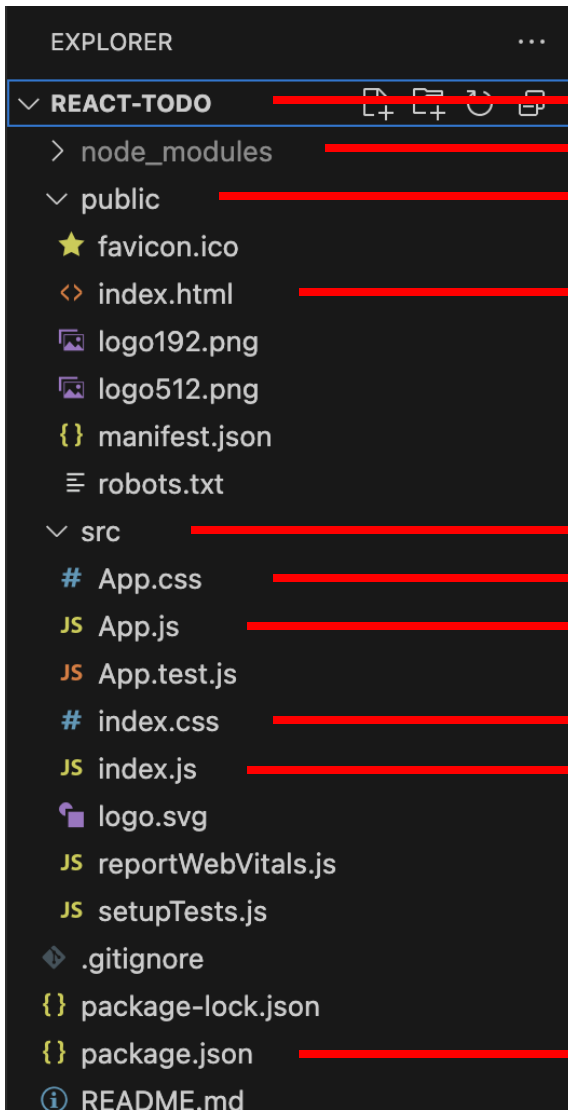
- Reactjs Project 생성
 - Command 창에서 아래와 같이 입력

```
> cd c:\W  
> mkdir react-projects  
> cd react-projects  
> npx create-react-app react-todo
```

- npx create-react-app 프로젝트명

3. 리액트 애플리케이션 생성

• 프로젝트 구성 알아보기



REACT-TODO : React Project 이름

node_modules : React Application 개발에 필요한 라이브러리들

public : React App에 필요한 Resource. 브라우저에서 접근 가능.

index.html : React App의 첫 페이지

src : React App을 구성하는 여러 파일들. 주로 컴포넌트가 위치함

App.css : App Component의 스타일 시트

App.js : React App의 메인(Root) 컴포넌트.

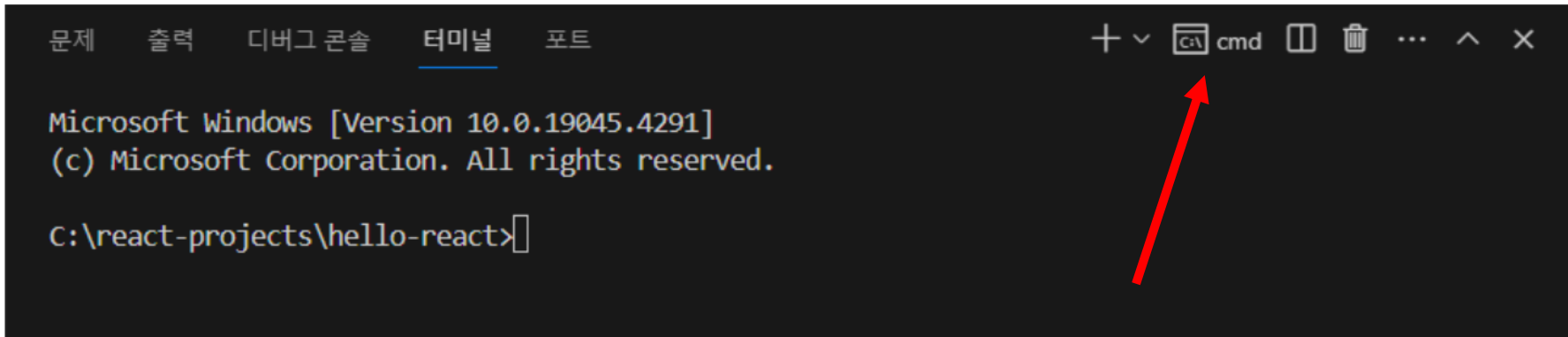
index.css : React App 전체의 스타일시트

index.js : React App을 구동시키는 Javascript

package.json : React Build, Dependency 등과 같은 설정을 하는 파일

3. 리액트 애플리케이션 생성

- React Application 실행시키기
 - 터미널 → 새 터미널



- 터미널이 “cmd”로 표시가 안된다면
 - 1. Ctrl + Shift + P → “터미널” 검색
 - 2. “터미널 : 기본프로필 선택” 클릭
 - 3. “Command Prompt” 선택

3. 리액트 애플리케이션 생성

- React Application 실행시키기
 - 터미널에 아래 명령어 입력 후 엔터
 - `npm run start`

Compiled successfully!

You can now view **hello-react** 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**



Edit `src/App.js` and save to reload.

[Learn React](#)

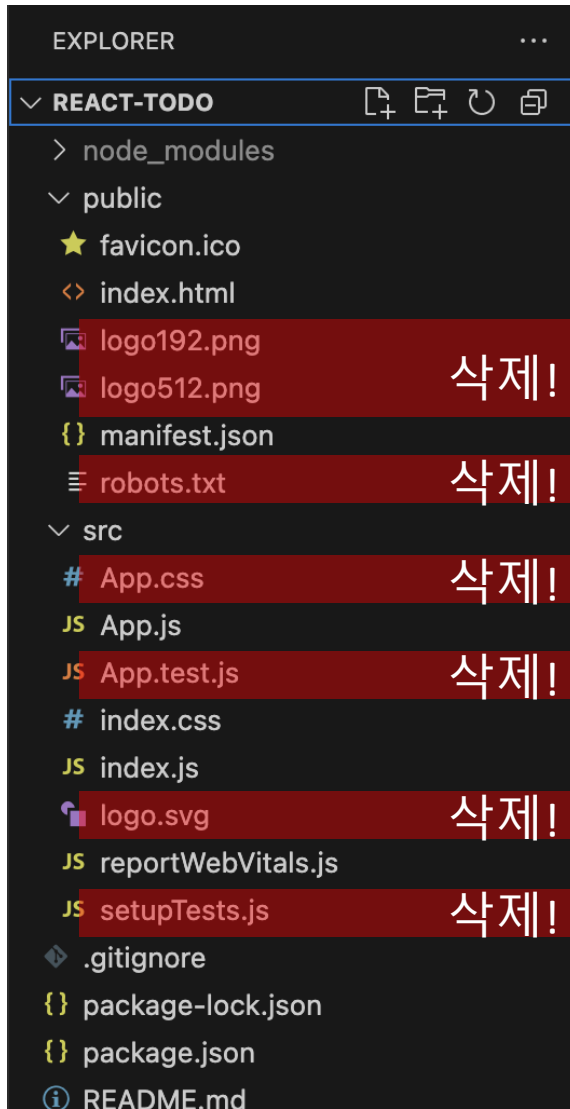
3. 리액트 애플리케이션 생성

- npm run start
- npm run은 package.json에 작성된 “scripts” 항목을 실행하는 명령어

```
14     "scripts": {  
15         "start": "react-scripts start",  
16         "build": "react-scripts build",  
17         "test": "react-scripts test",  
18         "eject": "react-scripts eject"  
19     },
```

3. 리액트 애플리케이션 생성

- 필요없는 파일 삭제



3. 리액트 애플리케이션 생성

- 실수로 `node_modules`를 지웠을 경우
 - `node_modules`는 React App의 Dependency Library를 모아둔 폴더
 - 만약 삭제한다면 React App은 정상 실행될 수 없음.
 - 실수로 지웠다면, 터미널에서 “`npm install`” 명령어를 실행.
- React Source를 Git, SVN 등에 공유하려면 `node_modules`를 제거해야함.
 - `node_modules`의 용량이 매우 커, 공유할 경우 시간이 오래걸림.

Component, JSX, props, state

React 핵심 요소

4. Component

5. JSX

6. props

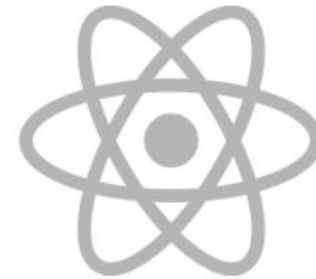
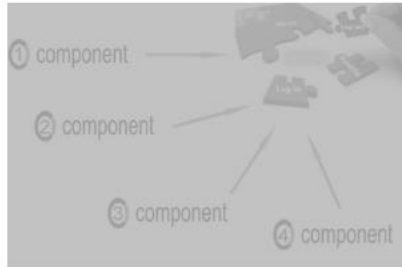
7. state



Component



Component-Based System



Component-Based
React Application

4. Component

4. Component

- 컴포넌트
 - 화면의 UI를 구성하는 함수
 - 블록 단위로 컴포넌트를 구성해, 코드를 단순하게 관리가능.
 - HTML의 Div, Section, Main, Article 등을 컴포넌트로 대체함
 - 컴포넌트를 재사용함으로써 코드 작성시간을 단축.



4. Component

- 리액트 컴포넌트
 - JSX문법으로 생성된 HTML Element(Tag) 에 대응하는 모듈
 - 모든 컴포넌트는 함수(function)로 작성한다.
- 리액트 컴포넌트 구분
 - 리액트에서 HTML Element는 첫 글자를 소문자로 작성
 - 리액트 컴포넌트는 첫 글자를 대문자로 작성
 - **컴포넌트의 첫 글자로 HTML Element와 컴포넌트를 구분한다.**

4. Component

컴포넌트의 이름은
대문자로 시작해야 한다.

컴포넌트는
함수로 정의한다.



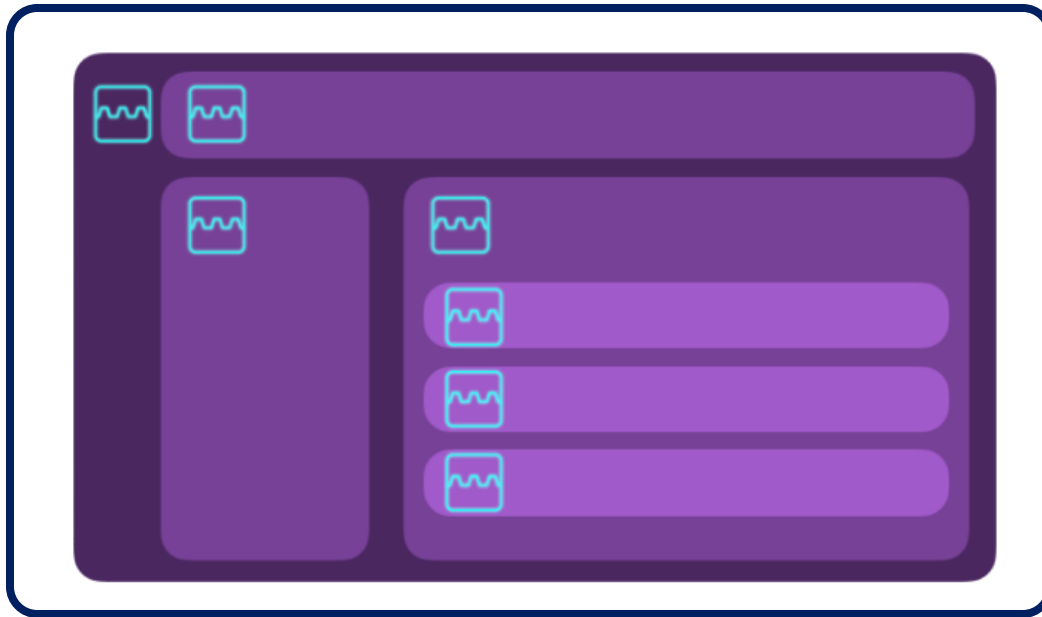
```
function App() {  
  return <div>This is a Component!</div>;  
}  
  
export default App;
```

컴포넌트를 외부에서 사용하려면
반드시 노출시켜야 한다.

4. Component

- React의 관례상, App.js의 App 컴포넌트는 Root Component가 된다.

App Component



- Root Component 는 index.js 에서 호출해야 함.

4. Component

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

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
```

```
  <React.StrictMode>
```

```
    <App />
```

```
  </React.StrictMode>
```

```
);
```

컴포넌트는 태그 형식으로
작성해 호출한다.

<App />은 App.js의 App 함수를 호출하고
함수가 반환하는 HTML코드가 화면에 노출된다.

4. Component

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

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
```

```
  <React.StrictMode>
```

```
    <App />
```

```
  </React.StrictMode>
```

```
);
```

컴포넌트는 태그 형식으로
작성해 호출한다.

<App />은 App.js의 App 함수를 호출하고
함수가 반환하는 HTML코드가 화면에 노출된다.

```
function App() {
  return <div>This is a Component!</div>;
}

export default App;
```

4. Component

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

```
const root =
ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
```

```
  <React.StrictMode>
```

```
    <App />
```

```
  </React.StrictMode>
```

```
);
```

컴포넌트는 태그 형식으로
작성해 호출한다.

<App />은 App.js의 App 함수를 호출하고
함수가 반환하는 HTML코드가 화면에 노출된다.

```
function App() {
  return <div>This is a Component!</div>;
}

export default App;
```


4. Component

- 실습: Todo Component 만들어보기 (1 / 2)

```
function App() {  
  return (  
    <div class="wrapper">  
      <header>React Todo</header>  
      <ul class="tasks">  
        <li class="tasks-header">  
          <input id="checkall" type="checkbox" />  
          <label>Task</label>  
          <span class="due-date">Due date</span>  
          <span class="priority">Priority</span>  
        </li>  
        <li class="task-item">  
          <input id="todo_1" type="checkbox" />  
          <label for="todo_1">React Component 마스크터</label>  
          <span class="due-date">2025-12-31</span>  
          <span class="priority">1</span>  
        </li>  
      </ul>  
    </div>  
  )  
}
```

4. Component

- 실습: Todo Component 만들어보기 (2 / 2)

```
<footer>
  <input type="text" placeholder="Task" />
  <input type="date" />
  <select>
    <option>우선순위</option>
    <option value="1">높음</option>
    <option value="2">보통</option>
    <option value="3">낮음</option>
  </select>
  <button type="button">Save</button>
</footer>
</div>
);
}

export default App;
```

4. Component

- Todo Component 확인

React Todo

<input type="checkbox"/> Task	Due date	Priority
<input type="checkbox"/> React Component 마스터	2025-12-31	1
<input type="checkbox"/> React Component 마스터	2025-12-31	1

연도. 월. 일. 

우선순위 

Save

5. JSX

5. JSX

- JSX 란,
 - XML같은 문법을 사용하는 ECMAScript의 확장.
 - 컴포넌트 렌더링을 구조화하는 방법을 제공한다.
- 컴포넌트를 구성하는 함수 내에서 HTML 코드와 Javascript 코드를 동시에 사용한다.

5. JSX

- JSX 빠르게 훑어보기
 - 1. JSX의 모든 태그는 종료 태그가 필수.
 - 2. JSX는 하나의 태그만 반환시켜야 한다.
 - `<Fragment> </Fragment>`
 - `<></>`
 - 3. 여러 줄의 태그를 반환할 경우 괄호 ()로 감싸야 한다.
 - (
 - `<>`
 - `<div> </div>`
 - `<div> </div>`
 - `</>`
 -);
 - 4. { } 를 이용해 변수나 상수, 함수를 바인딩 한다.
 - 변수, 상수, 함수, 스크립트, 주석 등을 작성

5. JSX

- JSX 빠르게 훑어보기
 - 5. Event Handling을 Camel Case로 작성한다.
 - html: onclick → React: onClick
 - html: onkeyup → React: onKeyUp
 - 6. class 지정은 className으로
 - 7. label for="" 는 label htmlFor="" 로 한다.
 - Javascript와 Class의 예약어 때문.

5. JSX

- 실습: Todo Component JSX 적용 (1 / 3)

```
function App() {  
  const onClickHandler = () => {  
    alert("Todo 추가!");  
  };  
  
  const onInputChangeHandler = (event) => {  
    console.log(event.currentTarget.value);  
  };  
  
  const onPriorityChangeHandler = (event) => {  
    alert(event.currentTarget.value);  
  };  
  
  return (  
    <div className="wrapper">  
      <header>React Todo</header>  
      <ul className="tasks">  
        <li className="tasks-header">  
          <input id="checkall" type="checkbox" />  
        </li>  
      </ul>  
    </div>  
  );  
}
```


5. JSX

- 실습: Todo Component JSX 적용 (2 / 3)

```
<label>Task</label>
<span className="due-date">Due date</span>
<span className="priority">Priority</span>
</li>
<li className="task-item">
  <input id="todo_1" type="checkbox" />
  <label htmlFor="todo_1">React Component 마스터</label>
  <span className="due-date">2025-12-31</span>
  <span className="priority">1</span>
</li>
</ul>
<footer>
  <input
    type="text"
    placeholder="Task"
    onKeyUp={onTodoInputKeyUpHandler}
  />
  <input type="date" />
  <select onChange={onPriorityChangeHandler}>
```

5. JSX

- 실습: Todo Component JSX 적용 (2 / 3)

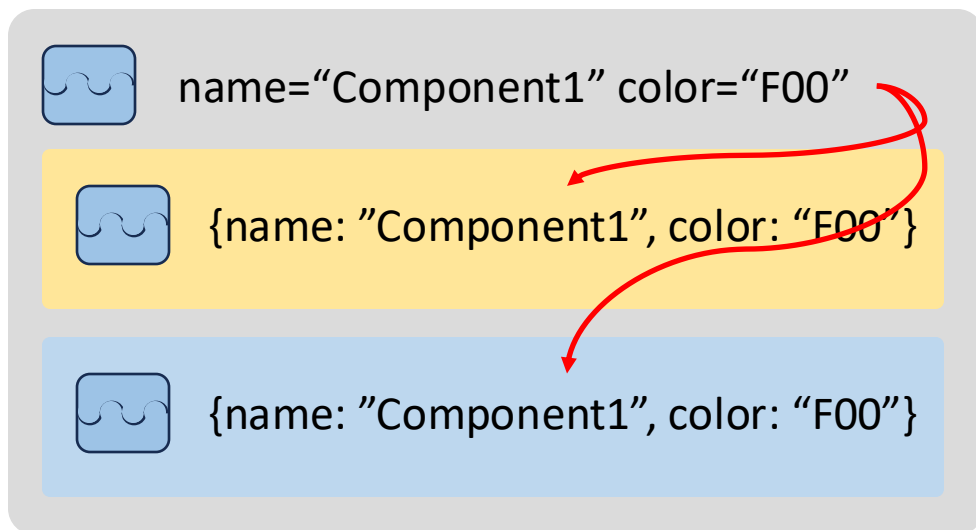
```
<option>우선순위</option>
<option value="1">높음</option>
<option value="2">보통</option>
<option value="3">낮음</option>
</select>
<button type="button" onClick={onButtonClickHandler}>
  Save
</button>
</footer>
</div>
);
}

export default App;
```

6. props

6. props

- props
 - 부모컴포넌트에서 자식컴포넌트로 데이터를 전송하는 방법.
 - 값, 변수, 상수, 함수 등을 컴포넌트로 전송할 수 있다.
 - 단, 상위 컴포넌트에서 하위 컴포넌트로만 전송할 수 있다.
 - 필요에 따라 HTML 태그 묶음 또는 컴포넌트도 전달 할 수 있다.

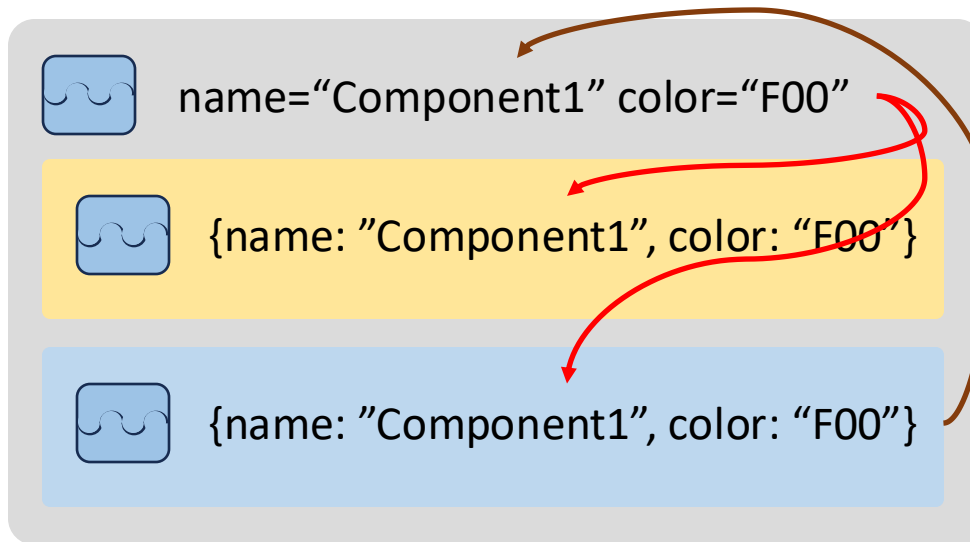


전달된 props는 모두 읽기 전용데이터!

6. props

- props

- 부모컴포넌트에서 자식컴포넌트로 데이터를 전송하는 방법.
- 값, 변수, 상수, 함수 등을 컴포넌트로 전송할 수 있다.
- 단, 상위 컴포넌트에서 하위 컴포넌트로만 전송할 수 있다.
- 필요에 따라 HTML 태그 묶음 또는 컴포넌트도 전달 할 수 있다.



❌ 자식컴포넌트에서
부모 컴포넌트로
Props를 전달할 수 없다.

전달된 props는 모두 읽기 전용데이터!

6. props

- props
 - 이미 보내진 Props의 내용과 새롭게 보내진 Props의 내용이 다를 경우, 컴포넌트를 다시 렌더링 한다.



name="Component1" color="F0F"

Props 변경감지!

컴포넌트를 다시 렌더링한다.



prevProps: {name: "Component1", color: "F00"}

newProps: {name: "Component1", color: "F0F"}

Props 변경감지!

컴포넌트를 다시 렌더링한다.



prevProps: {name: "Component1", color: "F00"}

newProps: {name: "Component1", color: "F0F"}

6. props

- App.js → Section.js

> /src/components/Section.js

```
export default function Section(props) {  
  const sectionCss = {  
    "backgroundColor": "#ccc",  
    color: props.color,  
  };  
  
  return (<div style={sectionCss}>  
    This is {props.title} Component  
  </div>);  
}
```

> /src/App.js

```
import Section from "../components/Section";  
  
function App() {  
  return <Section title="Section" color="#F00" />;  
}  
  
export default App;
```


6. props

- App.js → Section.js

> /src/components/Section.js

```
export default function Section(props) {  
  const sectionCss = {  
    "backgroundColor": "#ccc",  
    color: props.color,  
  };  
  
  return (<div style={sectionCss}>  
    This is {props.title} Component  
  </div>);  
}
```

props = {
 color: "#F00",
 title: "Section"
};



> /src/App.js

```
import Section from "../components/Section";  
  
function App() {  
  return <Section title="Section" color="#F00" />;  
}  
  
export default App;
```


6. props

- App.js → Section.js

> /src/components/Section.js

```
export default function Section(props) {  
  const sectionCss = {  
    "backgroundColor": "#ccc",  
    color: props.color,  
  };  
  
  return (<div style={sectionCss}>  
    This is {props.title} Component  
  </div>);  
}
```

props = {
 color: "#F00",
 title: "Section"
};

> /src/App.js

```
import Section from "../components/Section";  
  
function App() {  
  return <Section title="Section" color="#F00" />;  
}  
  
export default App;
```

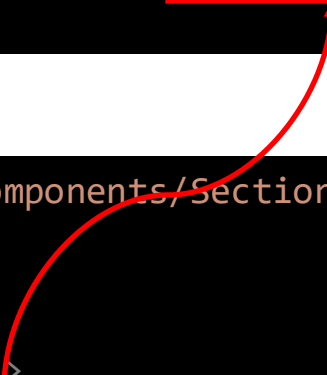
6. props

- App.js → Section.js (Tag 묶음 보내기)

> /src/components/Section.js

```
export default function Section(props) {  
  const sectionCss = {  
    "backgroundColor": "#ccc",  
    color: props.color,  
  };  
  
  return <div style={sectionCss}>{props.children}</div>;  
}
```

props = {
 color: "#F00",
 title: "Section"
 children: <div> ... </div>
};



> /src/App.js

```
import Section from "../components/Section";  
  
function App() {  
  return (  
    <Section color="#F00">  
      <div>This is a Section Components</div>  
    </Section>  
  );  
}  
  
export default App;
```

6. props

- App.js ➔ Section.js (여러개의 props 보내기)

> /src/components/Section.js

```
export default function Section(props) {
  const sectionCss = {
    "backgroundColor": props.backgroundColor,
    color: props.color,
    "fontSize": props.fontSize,
  };

  return <div style={sectionCss}>{props.children}</div>;
}
```

> /src/App.js

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

function App() {
  const fontColor = "#F00";
  return (
    <Section color={fontColor} backgroundColor="#FFF" fontSize="3rem">
      <div>This is a Section Components</div>
    </Section>
  );
}

export default App;
```

문자열 형태의 props는 따옴표로 감싸서 전송.
이외 형태(변수, 객체, 함수, 숫자, 불린 등)의 props는
중괄호로 감싸서 전송.

6. props

- App.js → Section.js (props를 구체적으로 전달 받기)

> /src/components/Section.js

```
export default function Section({
  color,
  backgroundColor,
  fontSize,
  children,
}) {
  const sectionCss = {
    "backgroundColor": backgroundColor,
    color: color,
    "fontSize": fontSize,
  };
  return <div style={sectionCss}>{children}</div>;
}
```

← props 객체를 분할해서 받아온다.

```
props = {
  color: "#F00",
  backgroundColor: "#FFF",
  fontSize: "3rem",
  children: <div>This ... Component </div>
};
```



App.js

6. props

- App.js → Section.js (함수 보내기)

> /src/components/Section.js

```
export default function Section({
  color,
  backgroundColor,
  fontSize,
  onMouseOver,
  children,
}) {
  const sectionCss = {
    backgroundColor: backgroundColor,
    color: color,
    fontSize: fontSize,
  };

  return (
    <div onMouseOver={onMouseOver} style={sectionCss}>
      {children}
    </div>
  );
}
```

props = {
 color: "#F00",
 backgroundColor: "#FFF",
 fontSize: "3rem",
 children: <div>This ... Component </div>
 onMouseOver: () => {console.log("...")}
};



App.js

6. props

- App.js → Section.js (함수 보내기)

> /src/App.js

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

function App() {
  const fontColor = "#F00";

  const hoverHandler = () => {
    console.log("Section 컴포넌트에 마우스가 오버되었음!");
  };

  return (
    <Section
      onMouseOver={hoverHandler}
      color={fontColor}
      backgroundColor="#FFF"
      fontSize="3rem"
    >
      <div>This is a Section Components</div>
    </Section>
  );
}

export default App;
```

6. props

- App.js → Section.js (함수 보내기)

```
function App() {  
  const hoverHandler = () => {  
    console.log("Section 컴포넌트에 마우스가 오버되었음!");  
  };  
}
```

```
return (  
  <Section  
    onMouseOver={hoverHandler}  
    ...  
  >...</Section>  
)  
);  
}
```

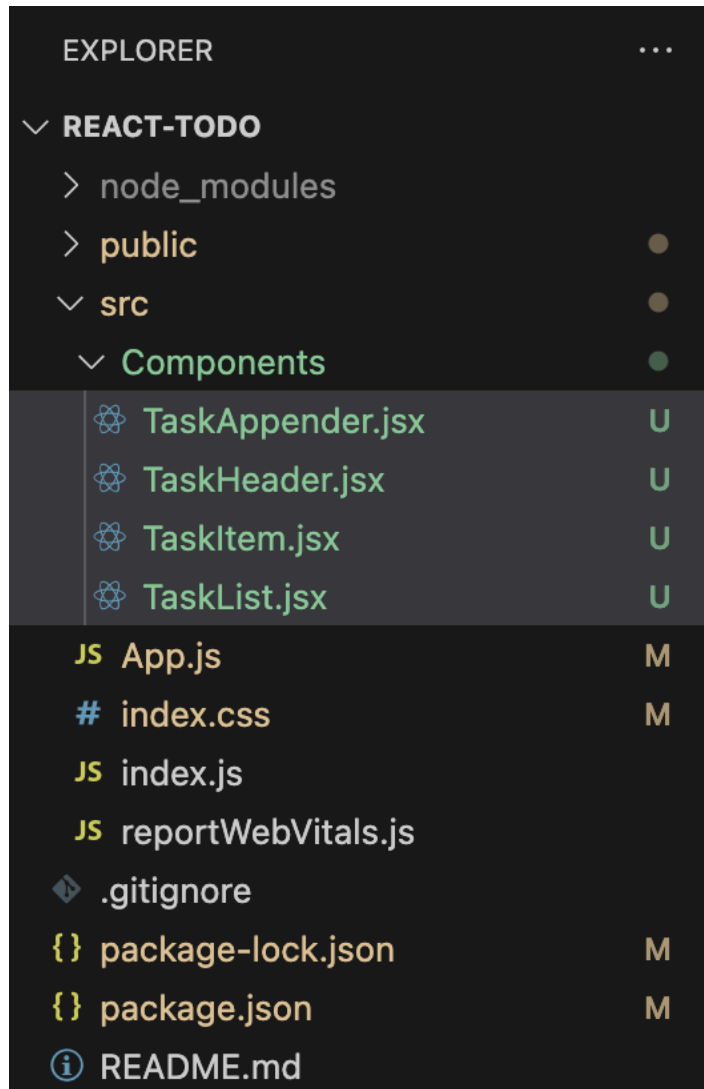
1. App.js 에서 선언된 함수(hoverHandler)가 Section 컴포넌트의 onMouseOver Props로 전달됨.

```
export default function Section({ ..., onMouseOver, children, }) {  
  ...  
  return (  
    <div onMouseOver={onMouseOver} style={sectionCss}>  
      {children}  
    </div>  
  );  
}
```

2. Section 컴포넌트의 div에서 mouseOver 이벤트가 발생하면, App.js가 보낸 함수가 실행됨.

6. props

- 실습: Todo Component 분리와 Props 전달



폴더 생성 및 파일 생성

6. props

- 실습: src > Components > TaskList.jsx

```
export default function TaskList({ children }) {  
  return <ul className="tasks">{children}</ul>;  
}
```

- 실습: src > Components > TaskHeader.jsx

```
export default function TaskHeader() {  
  return (  
    <li className="tasks-header">  
      <input id="checkall" type="checkbox" />  
      <label>Task</label>  
      <span className="due-date">Due date</span>  
      <span className="priority">Priority</span>  
    </li>  
  );  
}
```

6. props

- 실습: src > Components > TaskItem.jsx

```
export default function TaskItem({ id, task, dueDate, priority }) {  
  return (  
    <li className="task-item">  
      <input id={id} type="checkbox" />  
      <label htmlFor={id}>{task}</label>  
      <span className="due-date">{dueDate}</span>  
      <span className="priority">{priority}</span>  
    </li>  
  );  
}
```

6. props

- 실습: src > Components > TaskAppender.jsx (1 / 2)

```
export default function TaskAppender() {  
  const onClickHandler = () => {  
    alert("Todo 추가!");  
  };  
  
  const onKeyUpHandler = (event) => {  
    console.log(event.currentTarget.value);  
  };  
  
  const onChangeHandler = (event) => {  
    alert(event.currentTarget.value);  
  };  
  
  return (  
    <footer>  
      <input  
        type="text"  
        placeholder="Task"  
        onKeyUp={onKeyUpHandler} />  
    </footer>  
  );  
}
```

6. props

- 실습: src > Components > TaskAppender.jsx (2 / 2)

```
<input type="date" />
<select onChange={onPriorityChangeHandler}>
  <option>우선 순위</option>
  <option value="1">높음</option>
  <option value="2">보통</option>
  <option value="3">낮음</option>
</select>
<button type="button" onClick={onButtonClickHandler}>
  Save
</button>
</footer>
);
}
```

6. props

- 실습: src > App.js (1 / 2)

```
import TaskAppender from "../Components/TaskAppender";  
import TaskHeader from "../Components/TaskHeader";  
import TaskItem from "../Components/TaskItem";  
import TaskList from "../Components/TaskList";
```

```
function App() {  
  return (  
    <div className="wrapper">  
      <header>React Todo</header>  
      <TaskList>  
        <TaskHeader />  
        <TaskItem  
          id="item1"  
          task="React Component Master"  
          dueDate="2025-12-31"  
          priority="1"  
        />  
        <TaskItem  
          id="item2"
```

6. props

- 실습: src > App.js (2 / 2)

```
    task="React Props Master"  
    dueDate="2025-10-11"  
    priority="2"  
  />  
</TaskList>  
<TaskAppender />  
</div>  
);  
}  
  
export default App;
```

7. state

7. state

- 컴포넌트 내부에서 사용자가 발생시키는 이벤트에 따라 유동적으로 변경되어야 하는 변수/객체 등을 관리하는 방법.
 1. 사용자가 입력 박스에 키를 입력한다.
 2. 사용자가 Select 박스에서 값을 선택한다.
 3. 사용자가 Checkbox 의 아이템을 선택한다.
 4. 사용자가 Radio 의 아이템을 선택한다.
 5. 사용자가 Server에 새로운 데이터를 요청한다. 등.

7. state

- State 사용해보기

> /src/App.js (1 / 2)

```
import { useState } from "react";
import Section from "../components/Section";

function App() {
  const fontColor = "#F00";
  // title state를 만듦 (초기값은 "Untitled")
  // 이 시점에 title의 값은 "untitled".
  const [title, setTitle] = useState("Untitled");

  // input box에서 keyUp이벤트가 발생할 경우의 콜백
  const onKeyUpHandler = (event) => {
    // input box의 입력값을 받아옴.
    var value = event.currentTarget.value;
    // title state의 값을 변경
    // state는 setter를 통해 할당해야 한다!
    // 그렇지 않을 경우, 컴포넌트가 재실행 되지 않음.
    // 컴포넌트가 재실행 되어야만, 변경된 값으로 컴포넌트를 다시 그릴 수 있다.
    setTitle(value);
  };
}
```

7. state

- State 사용해보기

> /src/App.js (2 / 2)

```
// 컴포넌트가 재실행되는지 확인하기 위해 콘솔로그 작성
console.log(title);

return (
  <div>
    <input
      type="text"
      placeholder="컴포넌트 이름을 입력하세요."
      onKeyUp={onKeyUpHanlder}
    />
    <Section
      title={title}
      color={fontColor}
      backgroundColor="#FFF"
      fontSize="3rem"
    />
  </div>
);
}

export default App;
```

7. state

- State 사용해보기

> /src/components/Section.js

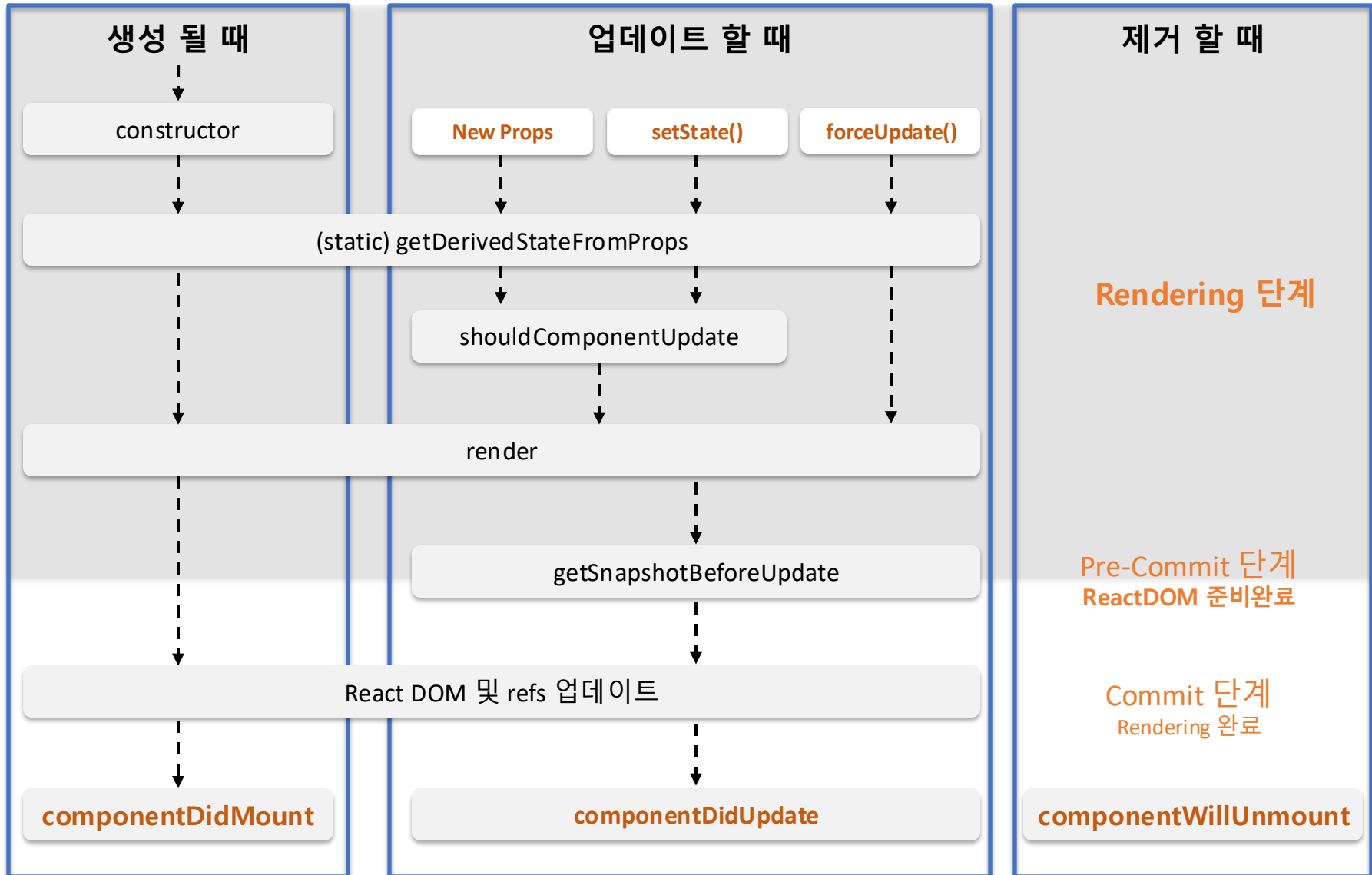
```
export default function Section({ title, color, backgroundColor, fontSize }) {  
  const sectionCss = {  
    backgroundColor: backgroundColor,  
    color: color,  
    fontSize: fontSize,  
  };  
  
  return <div style={sectionCss}>This is a {title} Component</div>;  
}
```

- App.js 에서 관리하는 title state가 Section으로 전달되려면,
- App 컴포넌트에서 Section 컴포넌트로 Props를 통해 전달되어야 한다.
- Section 입장에서 title은 state가 아닌 Props가 된다.

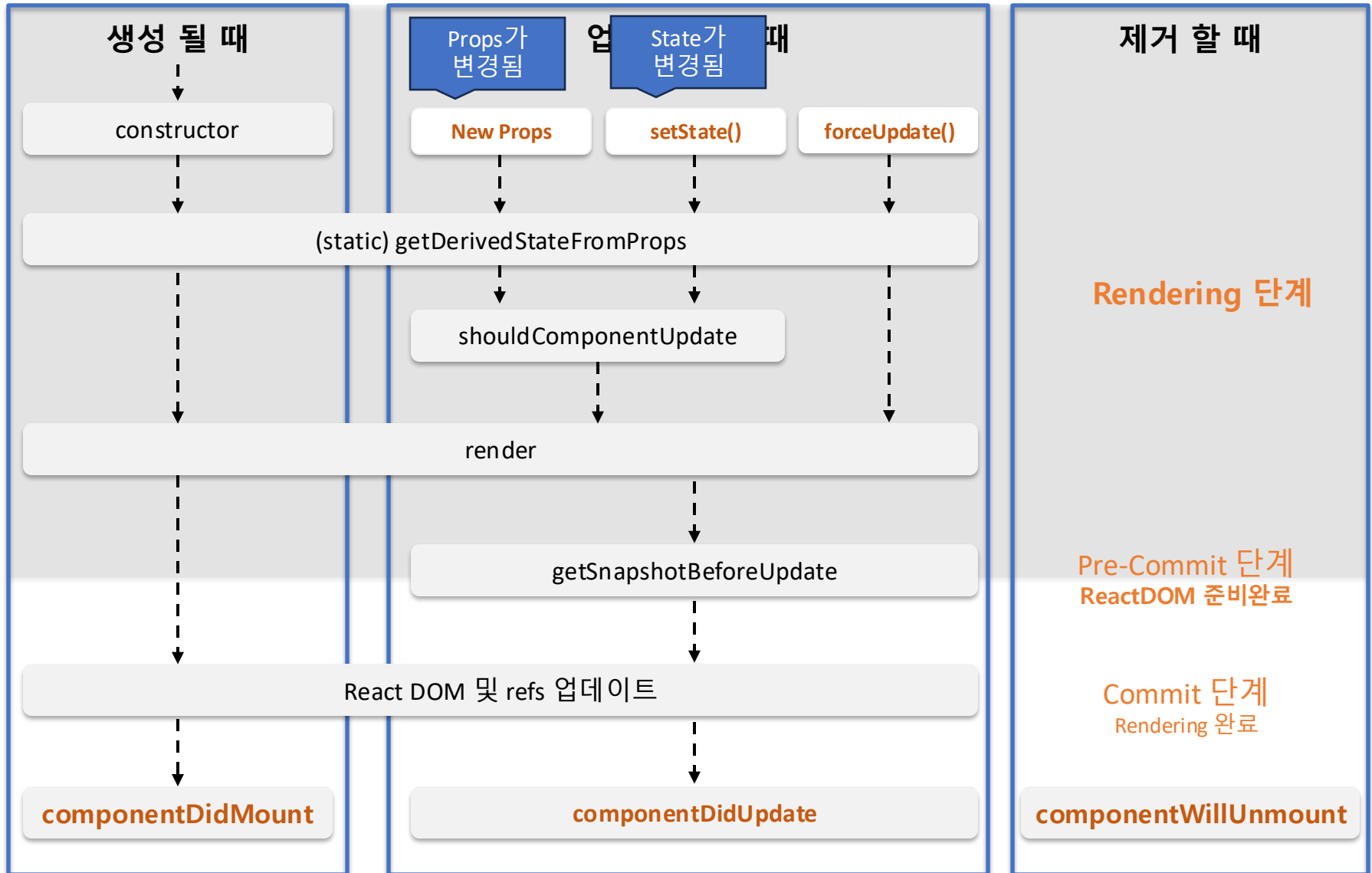
7. state

- Props와 State의 차이점.
 - Props는 부모 컴포넌트로 부터 받아오는 읽기 전용 데이터
 - State는 컴포넌트가 직접 관리하는 수정 가능한 데이터
- Props와 State의 공통점.
 - Props / State의 값이 변경될 경우, 이를 감지해 컴포넌트를 다시 실행시킨다.
 - 변경된 내용으로 브라우저에 보여주기 위함.

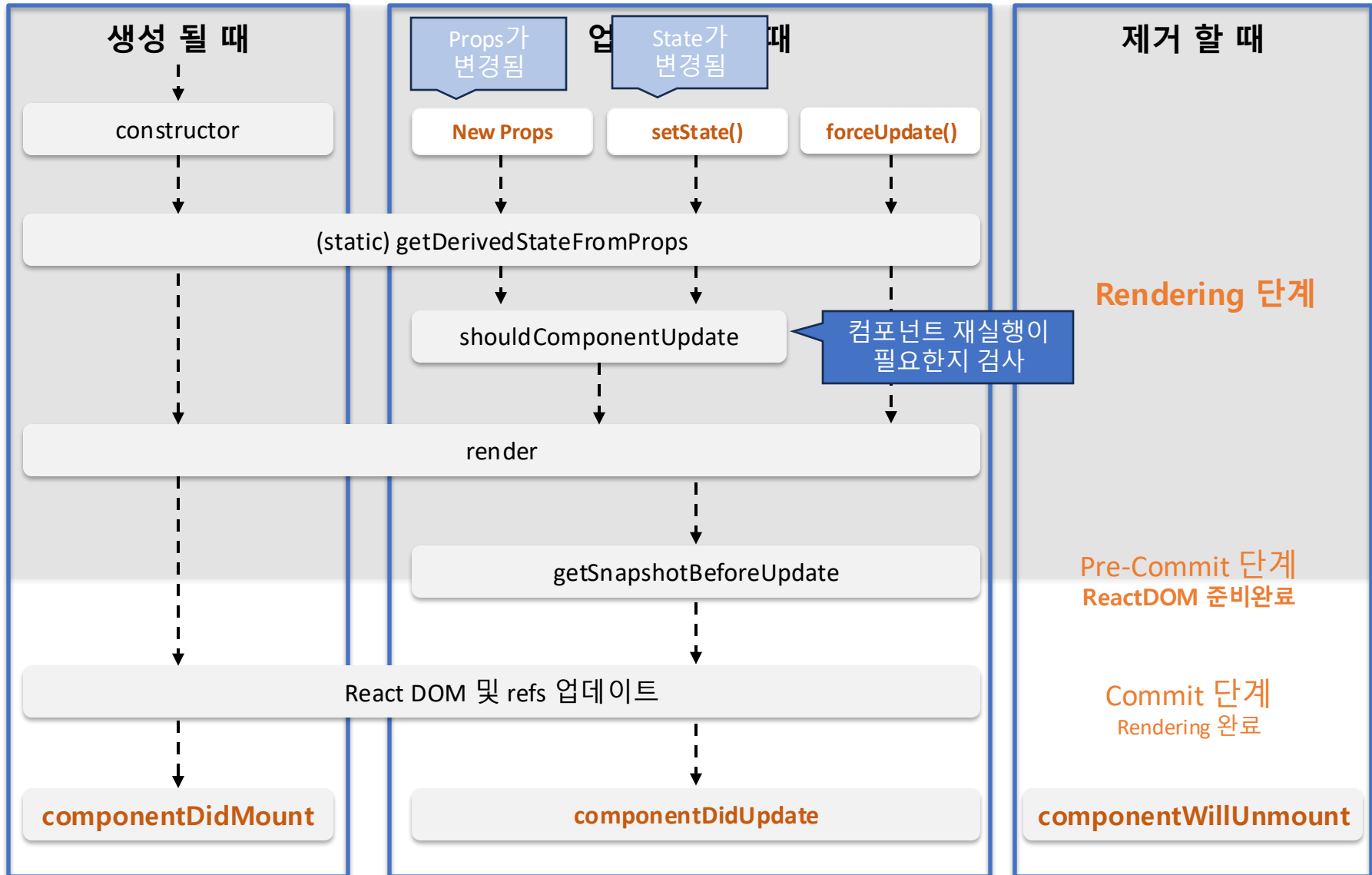
7. state (컴포넌트 Life-Cycle)



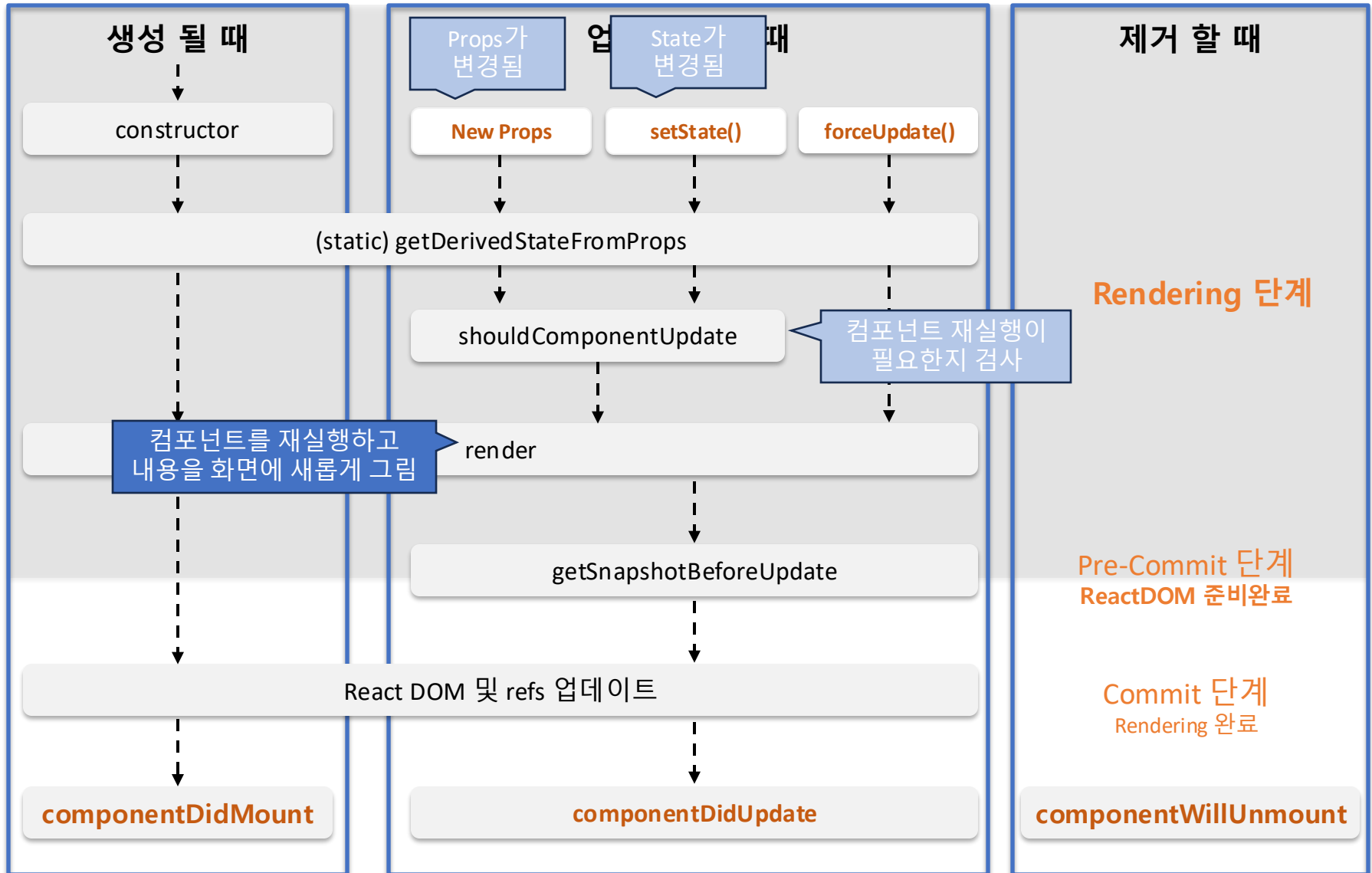
7. state (컴포넌트 Life-Cycle)



7. state (컴포넌트 Life-Cycle)



7. state (컴포넌트 Life-Cycle)



7. state

- 실습: Todo Component State 실습 및 화면 갱신
- 실습: src > App.js (1 / 7)

```
import { useState } from "react";
import TaskAppender from "../Components/TaskAppender";
import TaskHeader from "../Components/TaskHeader";
import TaskItem from "../Components/TaskItem";
import TaskList from "../Components/TaskList";

function App() {
  const [todoLists, setTodoList] = useState([
    {
      id: "item1",
      task: "React Component Master",
      dueDate: "2025-12-31",
      priority: 1,
      done: true,
    },
    {
      id: "item2",
```

7. state

- 실습: src > App.js (2 / 7)

```
    task: "React Props Master",  
    dueDate: "2025-10-11",  
    priority: 1,  
    done: true,  
  },  
  {  
    id: "item3",  
    task: "React States Master",  
    dueDate: "2025-09-07",  
    priority: 1,  
    done: false,  
  },  
]);
```

```
const addNewTodoHandler = (task, dueDate, priority) => {  
  setTodoList((prevTodoList) => {  
    const newTodoList = [...prevTodoList];  
    newTodoList.push({  
      id: "item" + (prevTodoList.length + 1),
```

7. state

- 실습: src > App.js (3 / 7)

```
task,  
  dueDate,  
  priority,  
  done: false,  
});  
return newTodoList;  
});  
};  
  
const doneTodoHandler = (event) => {  
  const todold = event.currentTarget.value;  
  if (  
    window.confirm(  
      `${todold} task를 완료할까요? 이 작업은 되돌릴 수 없습니다.`  
    )  
  ) {  
    setTodoList((prevTodoList) => {  
      const newTodoList = [...prevTodoList];
```

7. state

- 실습: src > App.js (4 / 7)

```
newTodoList.map((todo) => {  
  if (todo.id === todoId) {  
    todo.done = true;  
  }  
  return todo;  
});  
return newTodoList;  
});  
}  
};  
  
const doneAllTodoHandler = (event) => {  
  const processingTodoLength =  
    todoLists.filter((todo) => !todo.done).length;  
  if (event.currentTarget.checked && processingTodoLength === 0) {  
    alert("완료할 Task가 없습니다.");  
    event.currentTarget.checked = false;  
    return;  
  }  
}
```

7. state

- 실습: src > App.js (5 / 7)

```
if (
  event.currentTarget.checked &&
  window.confirm(
    "모든 task를 완료할까요? 이 작업은 되돌릴 수 없습니다.")
) {
  setTodoList((prevTodoList) => {
    const newTodoList = [...prevTodoList];

    newTodoList.map((todo) => {
      todo.done = true;
      return todo;
    });
    return newTodoList;
  });

  event.currentTarget.checked = false;
}
```

7. state

- 실습: src > App.js (6 / 7)

```
return (  
  <div className="wrapper">  
    <header>React Todo</header>  
    <TaskList>  
      <TaskHeader onClick={doneAllTodoHandler} />  
      {todoLists.map((item) => (  
        <TaskItem  
          key={item.id}  
          id={item.id}  
          task={item.task}  
          dueDate={item.dueDate}  
          priority={item.priority}  
          done={item.done}  
          onClick={doneTodoHandler}  
        />  
      ))}  
    </TaskList>  
    <TaskAppender onClick={addNewTodoHandler} />  
  </div>  
)
```

7. state

- 실습: src > App.js (7 / 7)

```
    </div>  
  );  
}  
  
export default App;
```

7. state

- 실습: src > Components/TaskAppender.jsx (1 / 3)

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

export default function TaskAppender({ onClick }) {
  const [task, setTask] = useState("");
  const [dueDate, setDueDate] = useState("");
  const [priority, setPriority] = useState();

  const onClickHandler = () => {
    if (!task) {
      alert("Task를 입력하세요.");
      return;
    }

    if (!dueDate) {
      alert("Due Date를 입력하세요.");
      return;
    }

    if (!priority) {
```


7. state

- 실습: src > Components/TaskAppender.jsx (2 / 3)

```
    alert("Priority를 선택하세요.");  
    return;  
  }  
  onClick(task, dueDate, priority);  
};  
  
const onTodoInputKeyUpHandler = (event) => {  
  setTask(event.currentTarget.value);  
};  
  
const onPriorityChangeHandler = (event) => {  
  setPriority(event.currentTarget.value);  
};  
  
const onDueDateChangeHandler = (event) => {  
  setDueDate(event.currentTarget.value);  
};  
  
return (
```

7. state

- 실습: src > Components/TaskAppender.jsx (3 / 3)

```
<footer>
  <input
    type="text"
    placeholder="Task"
    onKeyUp={onTodoInputKeyUpHandler} />
  <input type="date" onChange={onDueDateChangeHandler} />
  <select onChange={onPriorityChangeHandler}>
    <option value="">우선순위</option>
    <option value="1">높음</option>
    <option value="2">보통</option>
    <option value="3">낮음</option>
  </select>
  <button type="button" onClick={onButtonClickHandler}>
    Save
  </button>
</footer>
);
}
```

7. state

- 실습: src > Components/TaskHeader.jsx

```
export default function TaskHeader({ onChangeClick }) {  
  return (  
    <li className="tasks-header">  
      <input  
        id="checkall"  
        type="checkbox"  
        onChange={onChangeClick} />  
      <label>Task</label>  
      <span className="due-date">Due date</span>  
      <span className="priority">Priority</span>  
    </li>  
  );  
}
```

7. state

- 실습: src > Components/TaskItem.jsx (1 / 2)

```
export default function TaskItem({
  done,
  id,
  task,
  dueDate,
  priority,
  onCheckboxClick,
}) {
  return (
    <li className="task-item">
      <input
        id={id}
        type="checkbox"
        checked={done}
        value={id}
        onChange={onCheckboxClick}
        disabled={done}
      />
      <label
```

7. state

- 실습: src > Components/TaskItem.jsx (2 / 2)

```
htmlFor={id}
className={done ? "done-todo" : undefined}>
  {task}
</label>
<span
  className={`due-date ${done ? "done-todo" : undefined}`}>
  {dueDate}
</span>
<span
  className={`priority ${done ? "done-todo" : undefined}`}>
  {priority}
</span>
</li>
);
}
```

7. state

- 실습: src > Components/TaskList.jsx – 변경사항 없음.

```
export default function TaskList({ children }) {  
  return <ul className="tasks">{children}</ul>;  
}
```

Refs, Portals

8. Refs

9. forwardRef

10. useImperativeHandle

11. Portals

8. Refs

8. Refs

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

8. Refs

- DOM Focus를 지정하거나
- DOM Value를 참조하거나
- DOM 자체를 핸들링하거나(Modal 제어)
- DOM 이외의 캐시데이터가 필요할 경우, 유용하게 사용.

8. Refs

- Input 태그에 ref 연결.

> /src/App.js

```
import { useState, useRef } from "react";
import Section from "../components/Section";
```

```
function App() {
```

```
  ...
```

```
  const componentNameRef = useRef();
```

```
  ...
```

```
  return (
```

```
    <div>
```

```
      <input
```

```
        type="text"
```

```
        placeholder="컴포넌트 이름을 입력하세요."
```

```
        onKeyUp={onKeyUpHandler}
```

```
        ref={componentNameRef}
```

```
      />
```

```
    ...
```

```
  </div>
```

```
);
```

```
}
```

```
...
```

useRef를 선언하고
제어하고 싶은 DOM에
ref로 연결하기만 하면 끝!

8. Refs

- Input 태그에 ref 연결.

> /src/App.js

```
import { useState, useRef } from "react";
import Section from "../components/Section";

function App() {
  ...
  const onKeyUpHanlder = () => {
    //var value = event.currentTarget.value;
    console.log(componentNameRef); // ref 객체 출력
    console.log(componentNameRef.current); // ref에 연결된 dom 출력
    // ref에 연결된 dom의 value를 출력
    console.log(componentNameRef.current.value);
    setTitle(componentNameRef.current.value);
  };
  ...
}

export default App;
```

8. Refs

- State를 Ref로 변경하기
- 실습: src > Components/TaskAppender.jsx (1 / 4)

```
import { useRef /*, useState*/ } from "react";

export default function TaskAppender({ onClick }) {
  const taskRef = useRef();
  const dueDateRef = useRef();
  const priorityRef = useRef();

  // const [task, setTask] = useState("");
  // const [dueDate, setDueDate] = useState("");
  // const [priority, setPriority] = useState();

  const onClickHandler = () => {
    if (!taskRef.current.value) {
      alert("Task를 입력하세요.");
      taskRef.current.focus();
      return;
    }

    if (!dueDateRef.current.value) {
      alert("Due Date를 입력하세요.");
    }
  }
}
```

8. Refs

- State를 Ref로 변경하기
- 실습: src > Components/TaskAppender.jsx (2 / 4)

```
    dueDateRef.current.focus();
    return;
  }

  if (!priorityRef.current.value) {
    alert("Priority를 선택하세요.");
    priorityRef.current.focus();
    return;
  }

  onClick(
    taskRef.current.value,
    dueDateRef.current.value,
    priorityRef.current.value
  );
};

// const onTodoInputKeyUpHandler = (event) => {
//   setTask(event.currentTarget.value);
// };
```

8. Refs

- State를 Ref로 변경하기
- 실습: src > Components/TaskAppender.jsx (3 / 4)

```
// const onPriorityChangeHandler = (event) => {  
//   setPriority(event.currentTarget.value);  
// };  
  
// const onDueDateChangeHandler = (event) => {  
//   setDueDate(event.currentTarget.value);  
// };  
  
return (  
  <footer>  
    <input type="text" placeholder="Task" ref={taskRef} />  
    <input type="date" ref={dueDateRef} />  
    <select ref={priorityRef}>  
      <option value="">우선순위</option>  
      <option value="1">높음</option>  
      <option value="2">보통</option>  
      <option value="3">낮음</option>  
    </select>  
    <button type="button" onClick={onButtonClickHandler}>  
      Save
```

8. Refs

- State를 Ref로 변경하기
- 실습: src > Components/TaskAppender.jsx (4 / 4)

```
    </button>  
  </footer>  
);  
}
```


9. forwardRef

9. forwardRef

- Ref는 Props로 전달할 수 없는 객체.
- Ref를 자식컴포넌트로 전달하려면, 특수한 형태의 컴포넌트가 필요.
 - ➔ forwardRef Component
 - 2024년 4월 발표된 React 에서는 ref를 Props로 보낼 수 있도록 개선되었음.
- Ref / forwardRef로 Modal을 만들어보자!

9. forwardRef

- Modal Component 만들기

> /src/components/modal/Modal.js

```
export default function AlertModal({ children }) {  
  return (  
    <dialog className="modal">  
      <div className="modal-body">  
        <section className="modal-close-button">X</section>  
        {children}  
      </div>  
    </dialog>  
  );  
}
```

9. forwardRef

- Modal Component 만들기

> /src/App.js

```
...
import AlertModal from "../modal/Modal";

function App() {
  ...
  return (
    <div>
      ...
      <AlertModal>
        <div>
          <h3>컴포넌트의 이름을 입력하세요!</h3>
        </div>
      </AlertModal>
    </div>
  );
}

export default App;
```

9. forwardRef

- 컴포넌트의 이름을 입력하지 않은 상태로 [확인] 버튼을 클릭하면 AlertModal 실행시키기

> /src/App.js

```
...  
function App() {  
  ...  
  // const onKeyUpHanlder = () => {  
  // };  
  
  const onClickHandler = () => {  
    setTitle(componentNameRef.current.value);  
  };  
  
  return (  
    <div>  
      <input  
        ... // onKeyUp={onKeyUpHanlder}  
        ref={componentNameRef} />  
      <button onClick={onClickHandler}>확인</button>  
      ...  
    </div>  
  );  
}
```

9. forwardRef

- 컴포넌트의 이름을 입력하지 않은 상태로 [확인] 버튼을 클릭하면 AlertModal 실행시키기

> /src/App.js

```
...  
function App() {  
  ...  
  const [showAlertModal, setShowAlertModal] = useState(false);  
  ...  
  const onClickHandler = () => {  
    setTitle(componentNameRef.current.value);  
    setShowAlertModal(componentNameRef.current.value === "");  
  };  
  
  return (  
    <div>  
      ...  
      <AlertModal open={showAlertModal}>  
        ...  
      </AlertModal>  
    </div>  
  );  
}
```

9. forwardRef

- 컴포넌트의 이름을 입력하지 않은 상태로 [확인] 버튼을 클릭하면 AlertModal 실행시키기

> /src/components/modal/Modal.js

```
export default function AlertModal({ open, children }) {
  return (
    <dialog className="modal" open={open}>
      <div className="modal-body">
        <section className="modal-close-button">X</section>
        {children}
      </div>
    </dialog>
  );
}
```

- dialog 의 open
 - open 값이 true 일 경우, 다이얼로그를 노출시키고
 - open 값이 false 일 경우, 다이얼로그를 숨긴다.

9. forwardRef

- AlertModal 닫기 기능 만들기

> /src/App.js

```
...  
function App() {  
  ...  
  const onClickCloseAlertModalHandler = () => {  
    setShowAlertModal(false);  
  };  
  
  return (  
    <div>  
      ...  
      <AlertModal open={showAlertModal} onClose={onClickCloseAlertModalHandler}>  
        <div>  
          <h3>컴포넌트의 이름을 입력하세요!</h3>  
        </div>  
      </AlertModal>  
    </div>  
  );  
}  
  
export default App;
```


9. forwardRef

- AlertModal 닫기 기능 만들기

> /src/components/modal/Modal.js

```
export default function AlertModal({ open, onClose, children }) {  
  return (  
    <dialog className="modal" open={open}>  
      <div className="modal-body">  
        <section className="modal-close-button" onClick={onClose}>  
          X  
        </section>  
        {children}  
      </div>  
    </dialog>  
  );  
}
```

- AlertModal을 열고 닫기 위해서는 open, onClose props를 전달해야 함.
- Dialog modal은 dialog.showModal(), dialog.close() 로 제어해야함.
- Props로는 기능 제어가 불가능!

9. forwardRef

- AlertModal를 함수로 제어하자.

> /src/App.js (1 / 2)

```
import { useState, useRef } from "react";
import Section from "../components/Section";
import AlertModal from "../modal/Modal";

function App() {
  ...
  // const [showAlertModal, setShowAlertModal] = useState(false);
  ...
  const alertModalRef = useRef();

  const onClickHandler = () => {
    if (componentNameRef.current.value) {
      setTitle(componentNameRef.current.value);
    } else {
      alertModalRef.current.showModal();
    }
  };

  // const onClickCloseAlertModalHandler = () => {
  //   setShowAlertModal(false);
  // };
}
```

9. forwardRef

- AlertModal를 함수로 제어하자.

> /src/App.js (2 / 2)

```
return (  
  <div>  
    ...  
    <AlertModal ref={alertModalRef}>  
      <div>  
        <h3>컴포넌트의 이름을 입력하세요!</h3>  
      </div>  
    </AlertModal>  
  </div>  
>);  
>}
```

9. forwardRef

- AlertModal를 함수로 제어하자.

> /src/components/modal/Modal.js

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

const AlertModal = forwardRef(({ children }, ref) => {
  // export default function AlertModal({ open, onClose, children }) {
    const closeModal = () => {
      ref.current.close();
    };
    return (
      <dialog className="modal" ref={ref}>
        <div className="modal-body">
          <section className="modal-close-button" onClick={closeModal}>
            X
          </section>
          {children}
        </div>
      </dialog>
    );
  });

export default AlertModal;
```

forwardRef 컴포넌트는
부모컴포넌트에서 ref를
props로 받아들 수 있는
컴포넌트다.

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > Components/modal/Modal.jsx

```
export default function Alert({ ref, children }) {  
  const onClickCloseButton = () => {  
    ref.current.close();  
  };  
  
  return (  
    <dialog className="modal" ref={ref}>  
      <div className="modal-body">  
        <section  
          className="modal-close-button"  
          onClick={onClickCloseButton}>  
          X  
        </section>  
        {children}  
      </div>  
    </dialog>  
  );  
}
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > Components/modal/Confirm.jsx (1 / 2)

```
export default function Confirm({
  ref,
  children,
  okHandler = () => {},
  cancelHandler,
}) {
  const onClickCloseButton = () => {
    ref.current.close();
  };

  return (
    <dialog className="modal" ref={ref}>
      <div className="modal-body">
        {children}
        <section>
          <button
            type="button"
            className="confirm-ok"
            onClick={okHandler}>
            OK
          </button>
        </section>
      </div>
    </dialog>
  );
}
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > Components/modal/Confirm.jsx (2 / 2)

```
<button
  type="button"
  className="confirm-cancel"
  onClick={cancelHandler ?? onClickCloseButton}
>
  Cancel
</button>
</section>
</div>
</dialog>
);
}
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > Components/TaskAppender.jsx (1 / 3)

```
import { useRef, useState } from "react";
import Alert from "../modal/Modal";

export default function TaskAppender({ onClick }) {
  const taskRef = useRef();
  const dueDateRef = useRef();
  const priorityRef = useRef();
  const alertRef = useRef();

  const [modalMessage, setModalMessage] = useState();

  const onClickHandler = () => {
    if (!taskRef.current.value) {
      setModalMessage("Task를 입력하세요.");
      alertRef.current.showModal();
      return;
    }

    if (!dueDateRef.current.value) {
      setModalMessage("Due Date를 입력하세요.");
    }
  }
}
```


9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > Components/TaskAppender.jsx (2 / 3)

```
    alertRef.current.showModal();
    return;
  }

  if (!priorityRef.current.value) {
    setModalMessage("Priority를 선택하세요.");
    alertRef.current.showModal();
    return;
  }

  onClick(
    taskRef.current.value,
    dueDateRef.current.value,
    priorityRef.current.value
  );
};

return (
  <>
    <footer>
      <input type="text" placeholder="Task" ref={taskRef} />
    </footer>
  </>
);
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > Components/TaskAppender.jsx (3 / 3)

```
<input type="date" ref={dueDateRef} />
<select ref={priorityRef}>
  <option value="">우선순위</option>
  <option value="1">높음</option>
  <option value="2">보통</option>
  <option value="3">낮음</option>
</select>
<button type="button" onClick={onButtonClickHandler}>
  Save
</button>
</footer>
<Alert ref={alertRef}>
  <div>
    <h3>{modalMessage}</h3>
  </div>
</Alert>
</>
);
}
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > App.js (1 / 6)

```
import { useRef, useState } from "react";
import TaskAppender from "../Components/TaskAppender";
import TaskHeader from "../Components/TaskHeader";
import TaskItem from "../Components/TaskItem";
import TaskList from "../Components/TaskList";
import Confirm from "../Components/modal/Confirm";
import Alert from "../Components/modal/Modal";

function App() {
  const alertRef = useRef();
  const allDoneConfirmRef = useRef();
  const doneConfirmRef = useRef();

  const [allDoneConfirmMessage, setAllDoneConfirmMessage] = useState();
  const [alertMessage, setAlertMessage] = useState();
  const [todoLists, setTodoList] = useState([ -- 생략 -- ]);

  const addNewTodoHandler = (task, dueDate, priority) => {
    -- 생략 --
  };
}
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > App.js (2 / 6)

```
const doneTodoHandler = (event) => {  
  const todold = event.currentTarget.value;  
  setAllDoneConfirmMessage(  
    `${todold} task를 완료할까요? 이 작업은 되돌릴 수 없습니다.`  
  );  
  doneConfirmRef.current.showModal();  
  doneConfirmRef.todold = todold;  
};  
  
const doneTodoItemHandler = () => {  
  setTodoList((prevTodoList) => {  
    const newTodoList = [...prevTodoList];  
  
    newTodoList.map((todo) => {  
      if (todo.id === doneConfirmRef.todold) {  
        todo.done = true;  
      }  
      return todo;  
    });  
  });  
};
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > App.js (3 / 6)

```
    return newTodoList;
  });

  doneConfirmRef.current.close();
};

const doneAllTodoHandler = (event) => {
  const processingTodoLength = todoLists.filter(
    (todo) => !todo.done).length;
  if (event.currentTarget.checked && processingTodoLength === 0) {
    setAlertMessage("완료할 Task가 없습니다.");
    event.currentTarget.checked = false;
    alertRef.current.showModal();
    return;
  }

  if (event.currentTarget.checked) {
    event.currentTarget.checked = false;
    setAllDoneConfirmMessage(
      "모든 task를 완료할까요? 이 작업은 되돌릴 수 없습니다."
    );
  }
};
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > App.js (4 / 6)

```
);  
  
allDoneConfirmRef.current.showModal();  
}  
};  
  
const allDoneOkHandler = () => {  
  setTodoList((prevTodoList) => {  
    const newTodoList = [...prevTodoList];  
  
    newTodoList.map((todo) => {  
      todo.done = true;  
      return todo;  
    });  
    return newTodoList;  
  });  
  
  allDoneConfirmRef.current.close();  
};
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > App.js (5 / 6)

```
return (  
  <>  
    <div className="wrapper">  
      <header>React Todo</header>  
      <TaskList>  
        <TaskHeader onCheckboxClick={doneAllTodoHandler} />  
        {todoLists.map((item) => (  
          <TaskItem  
            key={item.id}  
            id={item.id}  
            task={item.task}  
            dueDate={item.dueDate}  
            priority={item.priority}  
            done={item.done}  
            onCheckboxClick={doneTodoHandler}  
          />  
        ))}  
      </TaskList>  
      <TaskAppender onButtonClick={addNewTodoHandler} />  
    </div>  
  )  
);
```

9. forwardRef

- Todo App에 Alert, Confirm 만들기
- 실습: src > App.js (6 / 6)

```
    </div>
    <Alert ref={alertRef}>
      <div>
        <h3>{alertMessage}</h3>
      </div>
    </Alert>
    <Confirm ref={allDoneConfirmRef} okHandler={allDoneOkHandler}>
      <div>{allDoneConfirmMessage}</div>
    </Confirm>
    <Confirm ref={doneConfirmRef} okHandler={doneTodoItemHandler}>
      <div>{allDoneConfirmMessage}</div>
    </Confirm>
  </>
);
}

export default App;
```


10. useImperativeHandle

10. useImperativeHandle

- Modal 컴포넌트를 개발하는 개발자의 성향에 따라
- 모달 태그가 달라질 수 있다.
 - A 개발자: <dialog>
 - B 개발자: <div>
- 모달의 태그에 따라 모달을 실행하는 방법이 달라진다.
- 이런 경우, useImperativeHandle을 이용해 함수를 통일시켜 ref를 통해 실행할 수 있다.

10. useImperativeHandle

- 모달 함수를 통일 시키자.

> /src/components/modal/Modal.js (1 / 2)

```
import { useRef, forwardRef, useImperativeHandle } from "react";

const AlertModal = forwardRef(({ children }, ref) => {
  const modal = useRef();

  useImperativeHandle(ref, () => {
    return {
      open() {
        modal.current.showModal();
      },
      close() {
        modal.current.close();
      },
    };
  });

  const closeModal = () => {
    modal.current.close();
  };
});
```

10. useImperativeHandle

- 모달 함수를 통일 시키자.

> /src/components/modal/Modal.js (1 / 2)

```
return (  
  <dialog className="modal" ref={modal}>  
    <div className="modal-body">  
      <section className="modal-close-button" onClick={closeModal}>  
        X  
      </section>  
      {children}  
    </div>  
  </dialog>  
>);  
});  
  
export default AlertModal;
```

10. useImperativeHandle

- 모달 함수를 통일 시키자.

> /src/App.js

```
...  
  
function App() {  
  ...  
  
  const onClickHandler = () => {  
    if (componentNameRef.current.value) {  
      setTitle(componentNameRef.current.value);  
    } else {  
      alertModalRef.current.open();  
    }  
  };  
  
  return (  
    ...  
  );  
}  
  
export default App;
```

10. useImperativeHandle

- useImperativeHandle로 forwardRef로 전달된 객체에게 일관된 함수를 제공할 수 있다.
- 이제 AlertModal은 dialog, div, p 등 어떤 형태로든 모달을 정의할 수 있고, 통일된 모달 함수를 통해 제어할 수 있게 되었다.



App.js

```
alertModalRef.current.open();
```

```
const alertModalRef = useRef();
```

```
<AlertModal ref={alertModalRef}>
```

```
...
```

```
</AlertModal>
```



AlertModal.js

```
const mRef = useRef();
```

```
useImperativeHandle(ref, () => {  
  return {  
    open() {mRef.current.showModal();}  
    close() {mRef.current.close();}  
  }  
});
```

```
<dialog ref={mRef}> ... </dialog>
```

10. useImperativeHandle

- useImperativeHandle로 forwardRef로 전달된 객체에게 일관된 함수를 제공할 수 있다.
- 이제 AlertModal은 dialog, div, p 등 어떤 형태로든 모달을 정의할 수 있고, 통일된 모달 함수를 통해 제어할 수 있게 되었다.



App.js

```
alertModalRef.current.open();

const alertModalRef = useRef();

<AlertModal ref={alertModalRef}>
...
</AlertModal>
```



AlertModal.js

```
const mRef = useRef();

useImperativeHandle(ref, () => {
  return {
    open() {mRef.current.showModal()}
    close() {mRef.current.close()}
  }
});

<dialog ref={mRef}> ... </dialog>
```

10. useImperativeHandle

- useImperativeHandle로 forwardRef로 전달된 객체에게 일관된 함수를 제공할 수 있다.
- 이제 AlertModal은 dialog, div, p 등 어떤 형태로든 모달을 정의할 수 있고, 통일된 모달 함수를 통해 제어할 수 있게 되었다.

 App.js

```
alertModalRef.current.open();
```

```
const alertModalRef = useRef();  
  
<AlertModal ref={alertModalRef}>  
...  
</AlertModal>
```

 AlertModal.js

```
const mRef = useRef();  
  
useImperativeHandle(ref, () => {  
  return {  
    open() {mRef.current.showModal();}  
    close() {mRef.current.close();}  
  }  
});  
  
<dialog ref={mRef}> ... </dialog>
```


10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > Components/modal/Confirm.jsx (1 / 2)

```
import { useImperativeHandle, useRef } from "react";
```

```
export default function Confirm({ -- 생략 -- }) {
```

```
  const confirmRef = useRef();
```

```
  useImperativeHandle(ref, () => {
```

```
    return {
```

```
      open() {
```

```
        confirmRef.current.showModal();
```

```
      },
```

```
      close() {
```

```
        confirmRef.current.close();
```

```
    },
```

```
  });
```

```
});
```

```
const onClickCloseButton = () => {
```

```
  ref.current.close();
```

```
};
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > Components/modal/Confirm.jsx (1 / 2)

```
return (  
  <dialog className="modal" ref={confirmRef}>  
    -- 생략 --  
  </dialog>  
);  
}
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > Components/modal/Modal.jsx (1 / 2)

```
import { useImperativeHandle, useRef } from "react";
```

```
export default function Alert({ ref, children }) {
```

```
  const alertRef = useRef();
```

```
  useImperativeHandle(ref, () => {
```

```
    return {
```

```
      open() {
```

```
        alertRef.current.showModal();
```

```
      },
```

```
      close() {
```

```
        alertRef.current.close();
```

```
    },
```

```
  });
```

```
});
```

```
const onClickCloseButton = () => {
```

```
  ref.current.close();
```

```
};
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > Components/modal/Modal.jsx (2 / 2)

```
return (  
  <dialog className="modal" ref={alertRef}>  
    -- 생략 --  
  </dialog>  
);  
}
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > Components/TaskAppender.jsx (1 / 2)

```
import { useRef, useState } from "react";
import Alert from "../modal/Modal";

export default function TaskAppender({ onClick }) {
  -- 생략 --

  const onClickHandler = () => {
    if (!taskRef.current.value) {
      setModalMessage("Task를 입력하세요.");
      alertRef.current.open();
      return;
    }

    if (!dueDateRef.current.value) {
      setModalMessage("Due Date를 입력하세요.");
      alertRef.current.open();
      return;
    }

    if (!priorityRef.current.value) {
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > Components/TaskAppender.jsx (2 / 2)

```
setModalMessage("Priority를 선택하세요.");  
alertRef.current.open();  
return;  
}  
-- 생략 --  
};  
  
return (  
  <>  
    -- 생략 --  
  </>  
);  
}
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > App.js (1 / 2)

-- 생략 --

```
function App() {
```

-- 생략 --

```
const doneTodoHandler = (event) => {  
  const todold = event.currentTarget.value;  
  setAllDoneConfirmMessage(  
    `${todold} task를 완료할까요? 이 작업은 되돌릴 수 없습니다.`  
  );  
  doneConfirmRef.current.open();  
  doneConfirmRef.todold = todold;  
};
```

-- 생략 --

```
const doneAllTodoHandler = (event) => {  
  const processingTodoLength = todoLists.filter(  
    (todo) => !todo.done).length;  
  if (event.currentTarget.checked && processingTodoLength === 0) {
```

10. useImperativeHandle

- Todo App에 Alert, Confirm 함수 만들기
- 실습: src > App.js (2 / 2)

```
    setAlertMessage("완료할 Task가 없습니다.");
    event.currentTarget.checked = false;
    alertRef.current.open();
    return;
  }

  if (event.currentTarget.checked) {
    event.currentTarget.checked = false;
    setAllDoneConfirmMessage(
      "모든 task를 완료할까요? 이 작업은 되돌릴 수 없습니다."
    );

    allDoneConfirmRef.current.open();
  }
};

-- 생략 --
}

export default App;
```


11. Portals

11. Portals

- 모달 같이 화면 최 상단에 나타나게 되는 DOM의 경우, BODY 태그의 가장 첫 번째 자식 요소가 되는 것이 유리.
- 보통, 동적으로 생성되는 DOM은 해당 컴포넌트를 기준으로 위치를 지정하게 된다.



11. Portals

- AlertModal의 위치를 <div id="root"> 위쪽으로 보내면, 관리가 더 수월해진다.
- 모달만 모아놓기 위해서 div#modals 생성.

> /public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="modals"></div>
    <div id="root"></div>
  </body>
</html>
```

11. Portals

- div#modals에 modal 보내기

> /src/App.js

```
...
import { createPortal } from "react-dom";

function App() {
  ...
  return (
    <div>
      ...
      {createPortal(
        <AlertModal ref={alertModalRef}>
          <div>
            <h3>컴포넌트의 이름을 입력하세요!</h3>
          </div>
        </AlertModal>,
        document.querySelector("#modals")
      )}
    </div>
  );
}
```

createPortal은 AlertModal 을
div#modal 내부로
이동시키는 역할을 한다.

```
export default App;
```

11. Portals

- div#modals에 modal 보내지는지 확인

Alert
Dialog

```
▼ <div id="modals">
  ▼ <dialog class="modal" open> top-layer (1) == $0
    ▶ <div class="modal-body"> ... </div> grid
      ::backdrop
    </dialog>
  </div>
  ▼ <div id="root">
    ▼ <div>
      <input type="text" placeholder="컴포넌트 이름을 입력하세요.">
      <button>확인</button>
      ▼ <div style="background-color: rgb(255, 255, 255); color: rgb(255, 0, 0); font-size: 3rem;">
        "This is a "
        "Untitled"
        " Component"
      </div>
    </div>
  </div>
```

11. Portals

- Todo App에 Alert, Confirm Portal 적용하기
- 실습: src > Components/modal/Modal.jsx (1 / 1)

```
import { useImperativeHandle, useRef } from "react";
import { createPortal } from "react-dom";

export default function Alert({ ref, children }) {
  -- 생략 --

  return createPortal(
    <dialog className="modal" ref={alertRef}>
      -- 생략 --
    </dialog>,
    document.querySelector("#modals")
  );
}
```

11. Portals

- Todo App에 Alert, Confirm Portal 적용하기
- 실습: src > Components/modal/Confirm.jsx (2 / 2)

```
import { useImperativeHandle, useRef } from "react";
import { createPortal } from "react-dom";

export default function Confirm({ -- 생략 -- }) {
  -- 생략 --

  return createPortal(
    <dialog className="modal" ref={confirmRef}>
      -- 생략 --
    </dialog>,
    document.querySelector("#modals")
  );
}
```

Context API

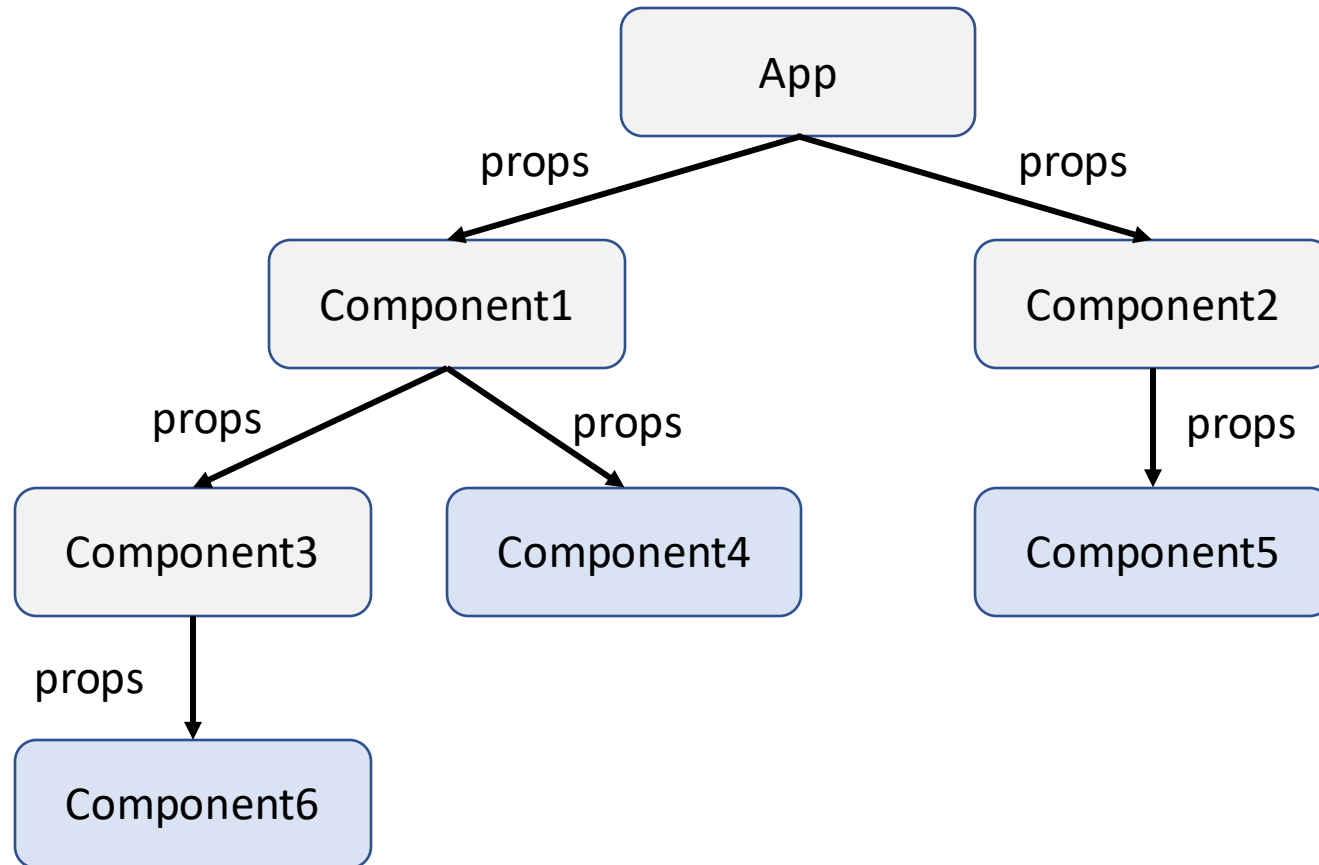
12. React Context

13. `useReducer`

12. React Context

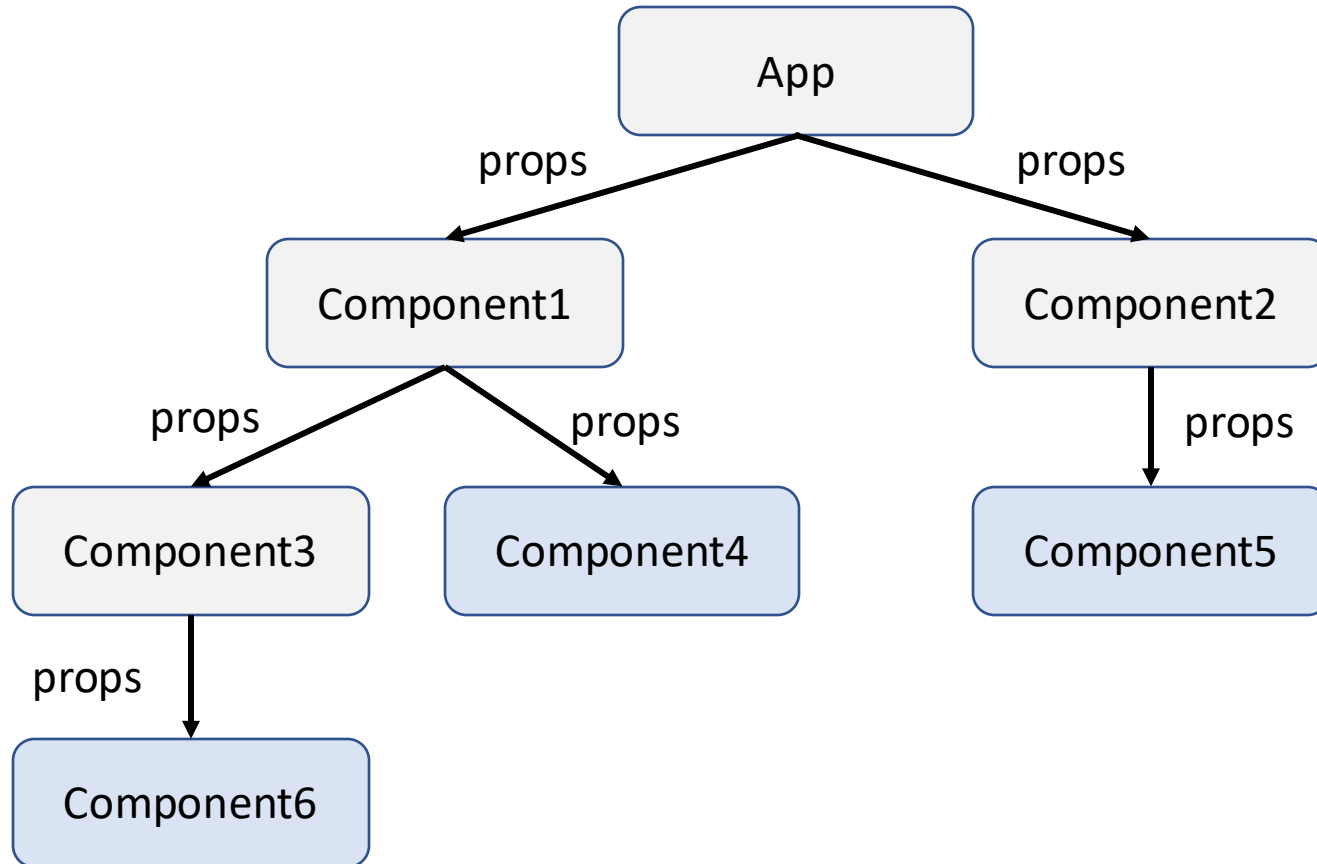
12. React Context

- App 에서 Component4, Component5, Component5 로 LoginData Props를 전달하려면..



12. React Context

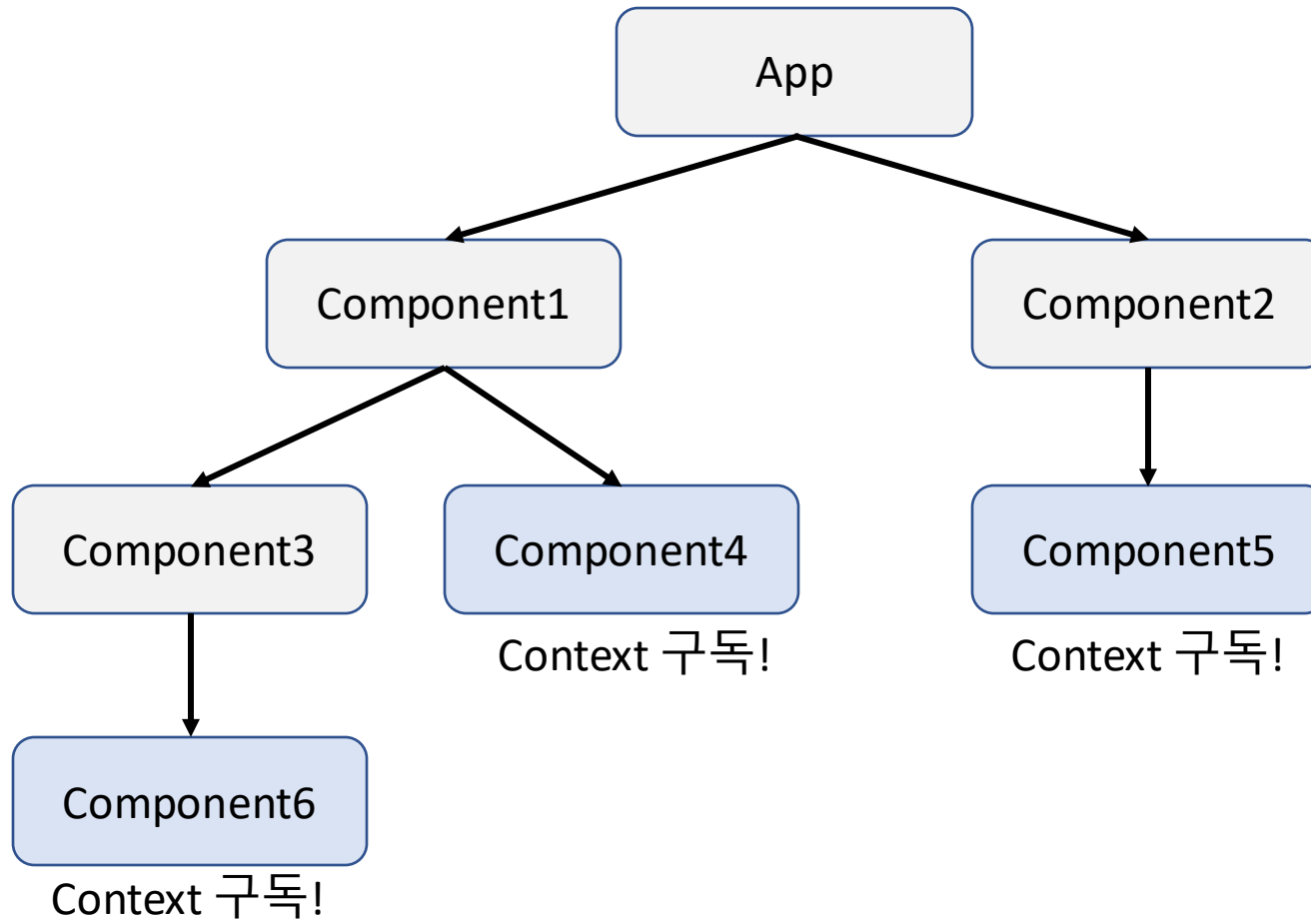
- App 에서 Component4, Component5, Component5 로 LoginData Props를 전달하려면..
- Props를 모두 전달해야만 한다. 만약 컴포넌트가 수십개라면?



12. React Context

- 만약, 컴포넌트마다 Props를 전달하지 않고 props가 필요한 컴포넌트에서만 불러 사용한다면?

Context 생성! – 공유 State를 한 곳에서 집중관리



12. React Context

- Todo App으로 Context 실습하기

React Todo

<input type="checkbox"/> Task	Due date	Priority
<input checked="" type="checkbox"/> React Component Master	2025-12-31	1
<input checked="" type="checkbox"/> React Props Master	2025-10-11	1
<input type="checkbox"/> React States Master	2025-09-07	1

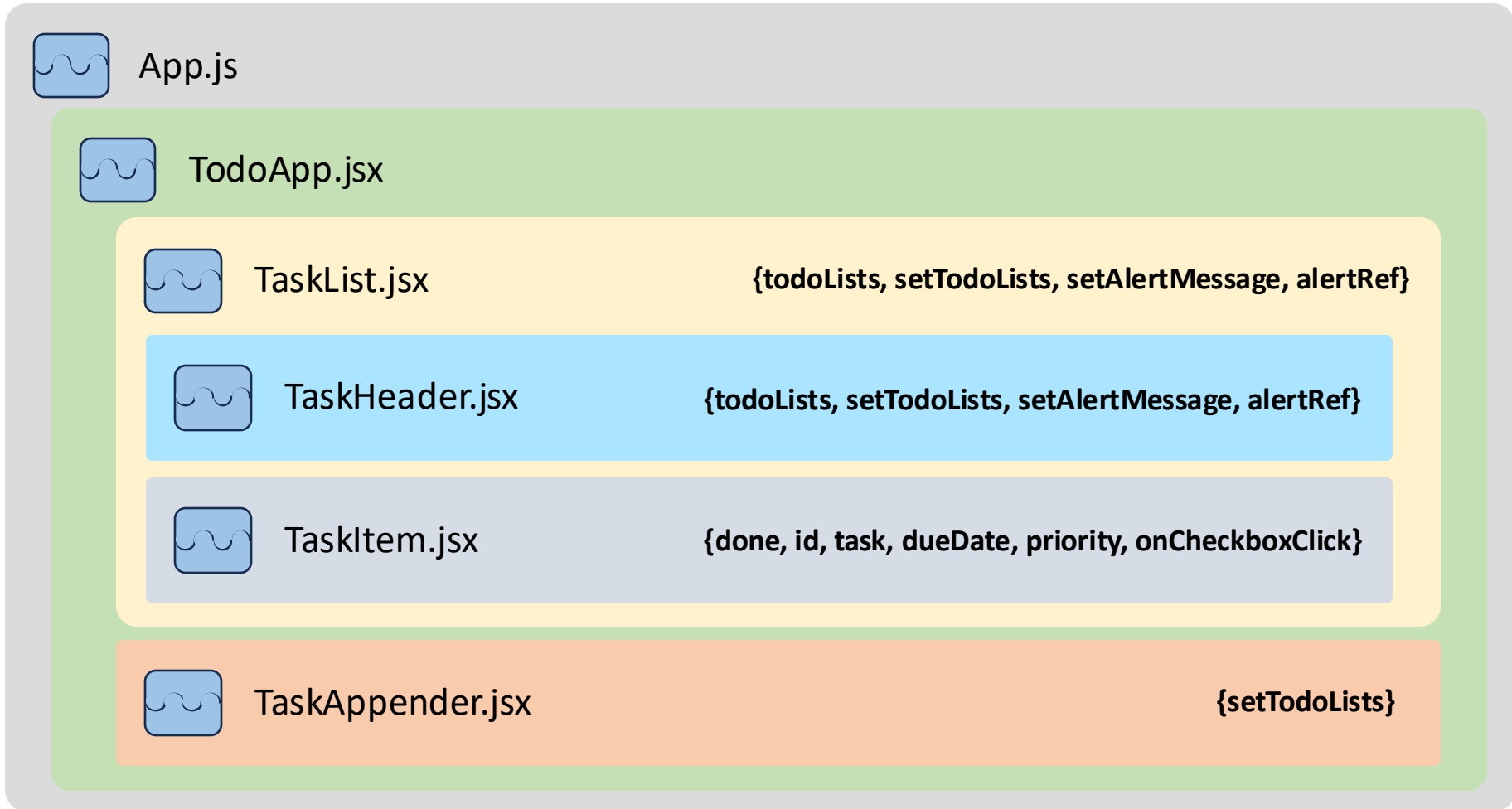
연도. 월. 일.

우선순위 ▼

Save

12. React Context

- Todo App 구조



12. React Context

- Prop Drilling (props를 필요한 컴포넌트까지 파고드는 컴포넌트 구조)
- Prop Drilling 은 가급적 피해야한다.
 - Props가 변경될 때마다 영향을 받는 컴포넌트는 재 실행되기 때문
 - 성능에 영향을 미친다.
- React Context는 Prop Drilling을 우회하게 해준다.
 - Prop Drilling 회피 목적으로만 Context를 사용하는 것은 비추천.
 - ➔ 이런 경우, 컴포넌트 합성을 추천.
 - Context의 값이 동적으로 변경되도록 해야한다.

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Contexts/TaskContext.jsx (1 / 3)

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

export const TaskContext = createContext({
  todoLists: [],
  done(taskId) {},
  allDone() {},
  add(task, dueDate, priority) {},
});

export default function TaskContextProvider({ children, initialState }) {
  const [todoLists, setTodoLists] = useState(initialState ?? []);

  const taskContextValue = {
    todoLists,
    done(taskId) {
      setTodoLists((prevTodoList) => {
        const newTodoList = [...prevTodoList];

        newTodoList.map((todo) => {
          if (todo.id === taskId) {
```


12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Contexts/TaskContext.jsx (2 / 3)

```
        todo.done = true;
      }
      return todo;
    });
    return newTodoList;
  });
},
allDone() {
  setTodoLists((prevTodoList) => {
    const newTodoList = [...prevTodoList];

    newTodoList.map((todo) => {
      todo.done = true;
      return todo;
    });
    return newTodoList;
  });
},
add(task, dueDate, priority) {
  setTodoLists((prevTodoList) => {
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Contexts/TaskContext.jsx (3 / 3)

```
const newTodoList = [...prevTodoList];
newTodoList.push({
  id: "item" + (prevTodoList.length + 1),
  task,
  dueDate,
  priority,
  done: false,
});
return newTodoList;
});
},
};

return (
  <TaskContext.Provider value={taskContextValue}>
    {children}
  </TaskContext.Provider>
);
}
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TodoApp.jsx (1 / 2)

```
import TaskList from "../TaskList";
import TaskAppender from "../TaskAppender";
import TaskContextProvider from "../Contexts/TaskContext";

export default function TodoApp() {
  const initialTasks = [
    {
      id: "item1",
      task: "React Component Master",
      dueDate: "2025-12-31",
      priority: 1,
      done: true,
    },
    {
      id: "item2",
      task: "React Props Master",
      dueDate: "2025-10-11",
      priority: 1,
      done: true,
    },
  ],
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TodoApp.jsx (2 / 2)

```
{
  id: "item3",
  task: "React States Master",
  dueDate: "2025-09-07",
  priority: 1,
  done: false,
},
];

return (
  <TaskContextProvider initialState={initialTasks}>
    <div className="wrapper">
      <header>React Todo</header>
      <TaskList />
      <TaskAppender />
    </div>
  </TaskContextProvider>
);
}
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기

> /src/Components /TaskList.jsx (1 / 3)

```
import { useContext, useRef, useState } from "react";
import TaskHeader from "../TaskHeader";
import TaskItem from "../TaskItem";
import Confirm from "../modal/Confirm";
import Alert from "../modal/Modal";
import { TaskContext } from "../../Contexts/TaskContext";

export default function TaskList() {
  const doneConfirmRef = useRef();
  const alertRef = useRef();

  const [alertMessage, setAlertMessage] = useState();
  const [doneConfirmMessage, setDoneConfirmMessage] = useState();

  const { todoLists, done } = useContext(TaskContext);

  const doneTodoHandler = (event) => {
    const todold = event.currentTarget.value;
    setDoneConfirmMessage(
      `${todold} task를 완료할까요? 이 작업은 되돌릴 수 없습니다.`
    );
  };
}
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TaskList.jsx (2 / 3)

```
);  
doneConfirmRef.current.open();  
doneConfirmRef.todoId = todold;  
};  
  
const doneTodoItemHandler = () => {  
  done(doneConfirmRef.todoId);  
  doneConfirmRef.current.close();  
};  
  
return (  
  <>  
    <ul className="tasks">  
      <TaskHeader setAlertMessage={setAlertMessage} alertRef={alertRef} />  
      {todoLists.map((item) => (  
        <TaskItem  
          key={item.id}  
          id={item.id}  
          task={item.task}  
          dueDate={item.dueDate}
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TaskList.jsx (3 / 3)

```
        priority={item.priority}
        done={item.done}
        onCheckboxClick={doneTodoHandler}
      />
    )}
  </ul>

  <Alert ref={alertRef}>
    <div>
      <h3>{alertMessage}</h3>
    </div>
  </Alert>
  <Confirm ref={doneConfirmRef} okHandler={doneTodoItemHandler}>
    <div>{doneConfirmMessage}</div>
  </Confirm>
</>
);
}
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기

> /src/Components /TaskHeader.jsx (1 / 2)

```
import { useContext, useRef, useState } from "react";
import Confirm from "../modal/Confirm";
import { TaskContext } from "../Contexts/TaskContext";

export default function TaskHeader({ setAlertMessage, alertRef }) {
  const allDoneConfirmRef = useRef();

  const [allDoneConfirmMessage, setAllDoneConfirmMessage] = useState();
  const { todoLists, allDone } = useContext(TaskContext);

  const doneAllTodoHandler = (event) => {
    -- 생략 --
  };

  const allDoneOkHandler = () => {
    allDone();
    allDoneConfirmRef.current.close();
  };

  return (
```


12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TaskHeader.jsx (2 / 2)

```
<>
  -- 생략 --
</>
);
}
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TaskAppender.jsx (1 / 3)

```
import { useContext, useRef, useState } from "react";
import Alert from "../modal/Modal";
import { TaskContext } from "../Contexts/TaskContext";

export default function TaskAppender() {
  const taskRef = useRef();
  const dueDateRef = useRef();
  const priorityRef = useRef();
  const alertRef = useRef();

  const { add } = useContext(TaskContext);
  const [modalMessage, setModalMessage] = useState();

  const onClickHandler = () => {
    if (!taskRef.current.value) {
      setModalMessage("Task를 입력하세요.");
      alertRef.current.open();
      return;
    }
  }
```

12. React Context

- React Context를 이용해 Prop Drilling 끊기

> /src/Components/TaskAppender.jsx (2 / 3)

```
if (!dueDateRef.current.value) {  
  setModalMessage("Due Date를 입력하세요.");  
  alertRef.current.open();  
  return;  
}
```

```
if (!priorityRef.current.value) {  
  setModalMessage("Priority를 선택하세요.");  
  alertRef.current.open();  
  return;  
}
```

```
add(  
  taskRef.current.value,  
  dueDateRef.current.value,  
  priorityRef.current.value  
);  
};
```

```
return (  
<>
```

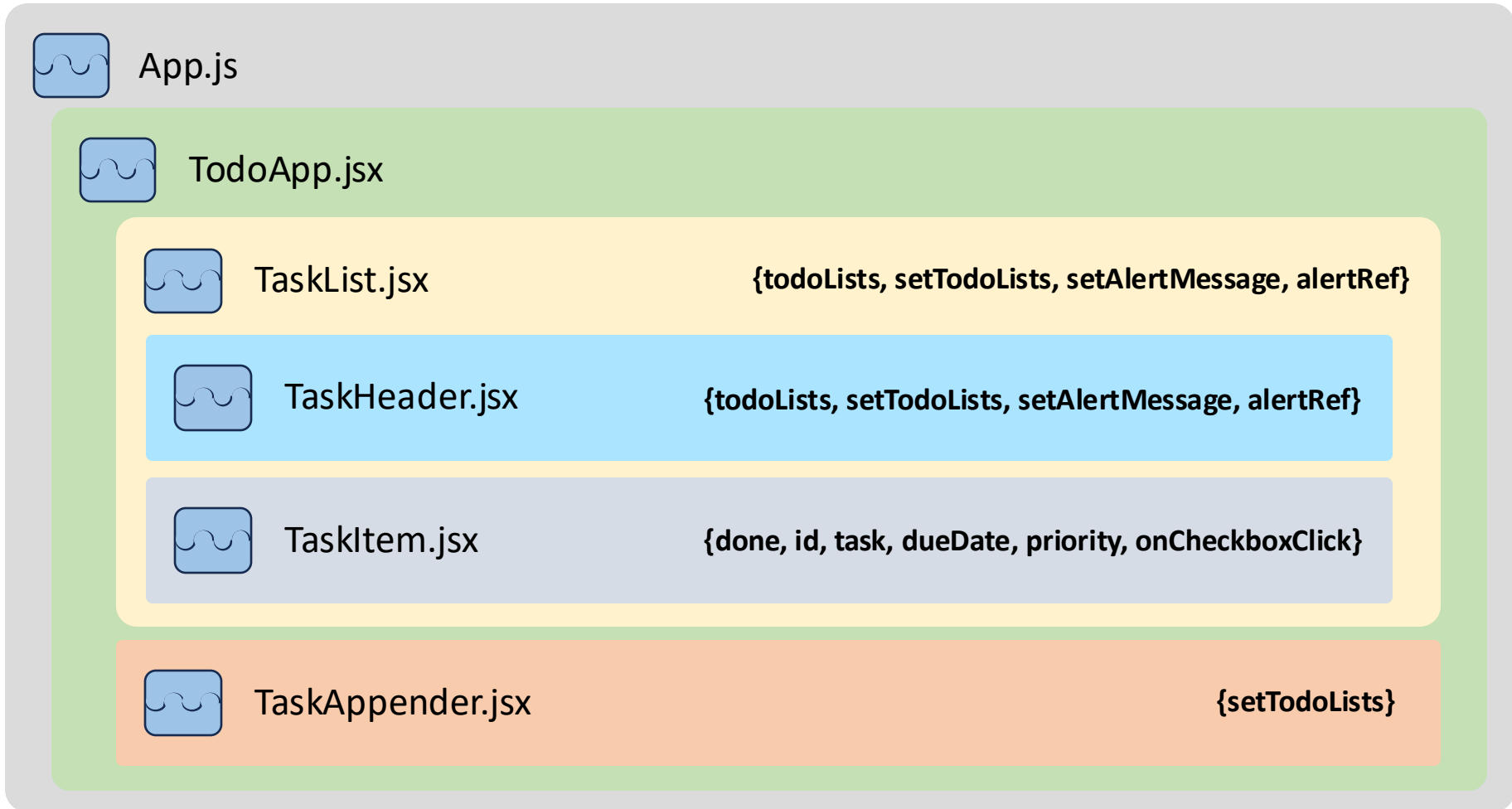
12. React Context

- React Context를 이용해 Prop Drilling 끊기
 - > /src/Components /TaskAppender.jsx (3 / 3)

```
-- 생략 --  
</>  
);  
}
```

12. React Context

- Todo App 구조 (Context 적용 이전)



12. React Context



App.js



TodoApp.jsx



TodoContext.Provider

State, setState 제공



TaskList.jsx



TaskHeader.jsx

{setAlertMessage, alertRef}



TaskItem.jsx

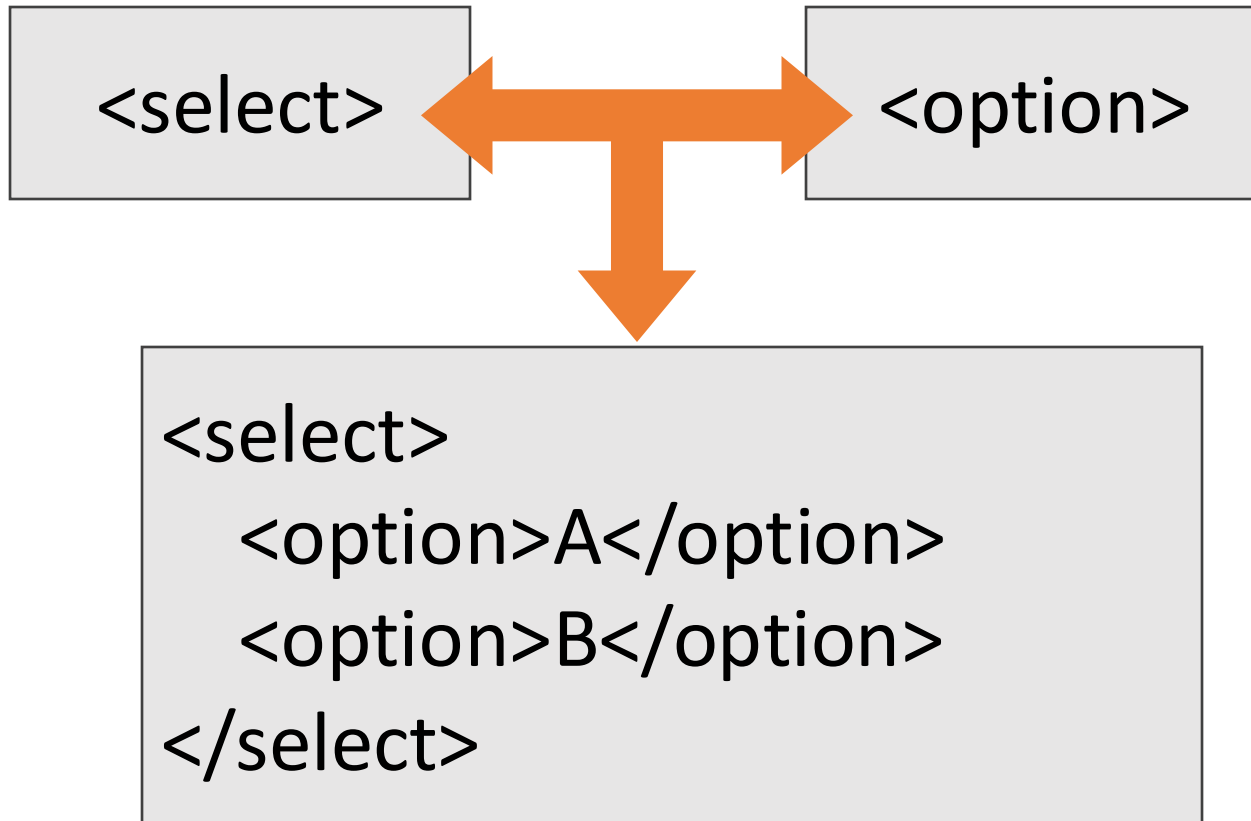
{done, id, task, dueDate, priority, onCheckboxClick}



TaskAppender.jsx

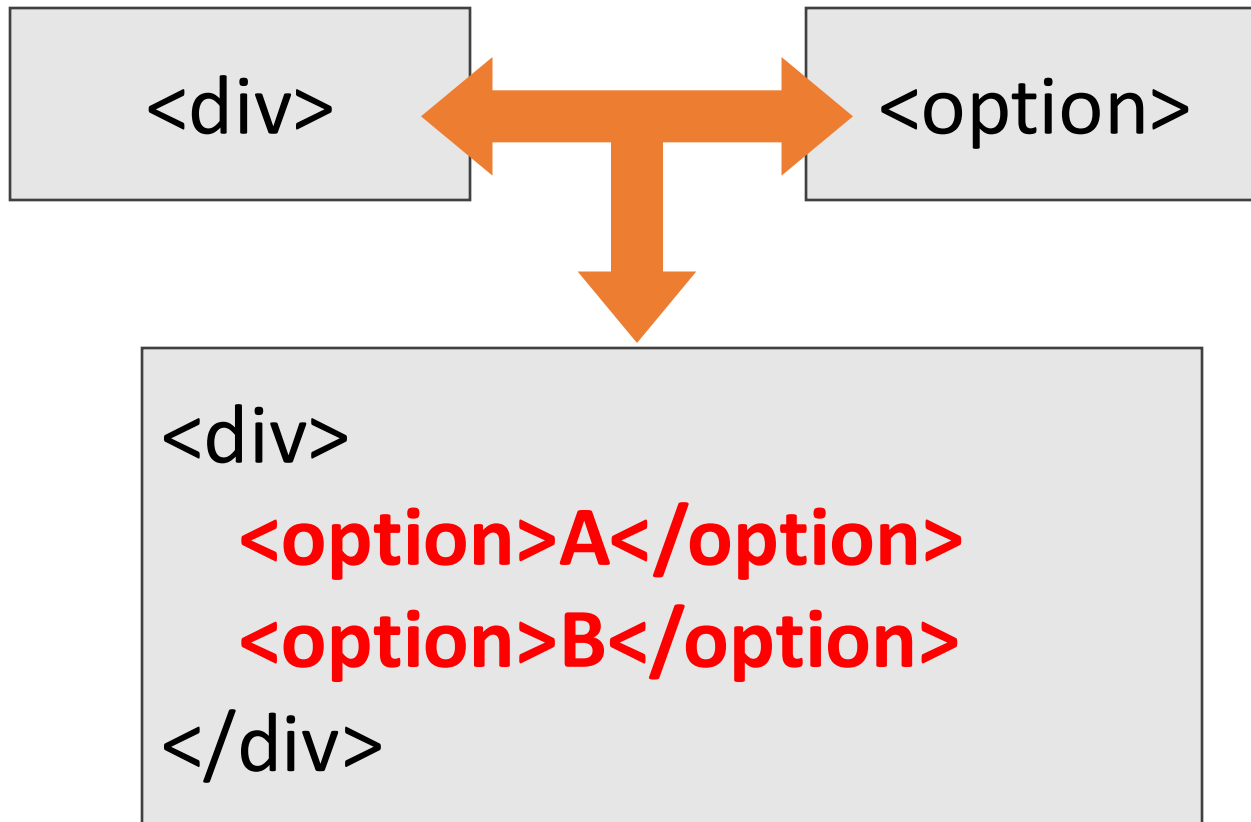
12. React Context

- 컴포넌트 합성?
 - 단일 컴포넌트(태그)는 아무런 역할을 할 수 없지만
 - 여러 컴포넌트(태그)와 조합하면 역할을 수행하는 컴포넌트(태그)



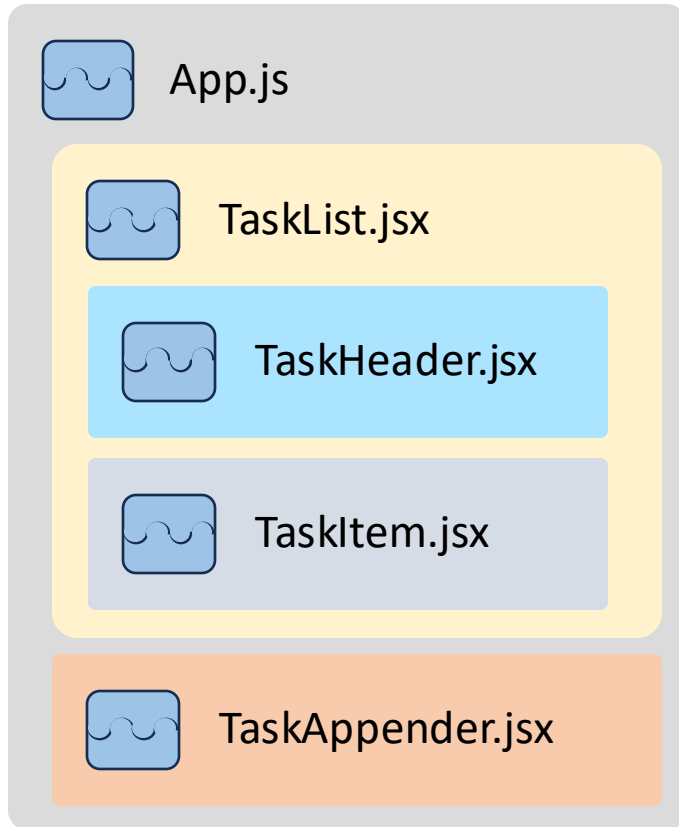
12. React Context

- `<option>` 태그는 올바른 위치에서만 동작하며
- 올바른 위치가 아닐 경우 “에러”가 발생한다.



12. React Context

- React Component 또한 여러 컴포넌트들을 합성할 수 있다.
- 사용 위치를 강제하기 위해서 Context가 사용된다. (**react-toto 에서**)



TaskHeader & TaskItem
TaskList 내부에서만 사용가능

12. React Context

- TaskHeader 와 TaskItem 내용을 TaskList로 이동.
 - TaskHeader & TaskItem 은 export 하지 않음.
- 이동 후 TaskHeader.jsx, TaskItem.jsx 파일 삭제.

12. React Context

> /src/Components /TaskList.jsx

```
export default function TaskList({ children }) {  
  return <ul className="tasks">{children}</ul>;  
}  
  
function TaskHeader({ onCheckboxClick }) {  
  return (  
    <li className="tasks-header">  
      -- 생략 --  
    </li>  
  );  
}  
  
function TaskItem({ done, id, task, dueDate, priority, onCheckboxClick }) {  
  return (  
    <li className="task-item">  
      -- 생략 --  
    </li>  
  );  
}  
  
TaskList.TaskHeader = TaskHeader;  
TaskList.TaskItem = TaskItem;
```

12. React Context

- 컴포넌트 합성을 위한 Context 생성

> /src/Components /TaskList.jsx

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

const TaskListContext = createContext();

export default function TaskList({ children }) {
  const contextValue = {componentName: "TaskList"};
  return (
    <TaskListContext.Provider value={contextValue}>
      <ul className="tasks">{children}</ul>
    </TaskListContext.Provider>
  );
}
-- 생략 --
```

12. React Context

- Context로 합성 컴포넌트 관리하기
 - > /src/Components /TaskList.jsx > TaskHeader

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

-- 생략 --

function TaskHeader({ onCheckboxClick }) {
  const context = useContext(TaskListContext);
  if (!context) {
    throw new Error("TaskHeader 컴포넌트는 TaskList 내부에 위치해야 합니다.");
  }
  return (
    <li className="tasks-header">
      <input id="checkall" type="checkbox" onChange={onCheckboxClick} />
      <label>Task</label>
      <span className="due-date">Due date</span>
      <span className="priority">Priority</span>
    </li>
  );
}
```

-- 생략 --

12. React Context

- Context로 합성 컴포넌트 관리하기

> /src/Components /TaskList.jsx > TaskItem

```
-- 생략 --  
function TaskItem({ done, id, task, dueDate, priority, onCheckboxClick }) {  
  const context = useContext(TaskListContext);  
  if (!context) {  
    throw new Error("TaskItem 컴포넌트는 TaskList 내부에 위치해야 합니다.");  
  }  
  return (  
    -- 생략 --  
  );  
}
```

```
-- 생략 --
```

12. React Context

- Context로 합성 컴포넌트 관리하기

> /src/App.js

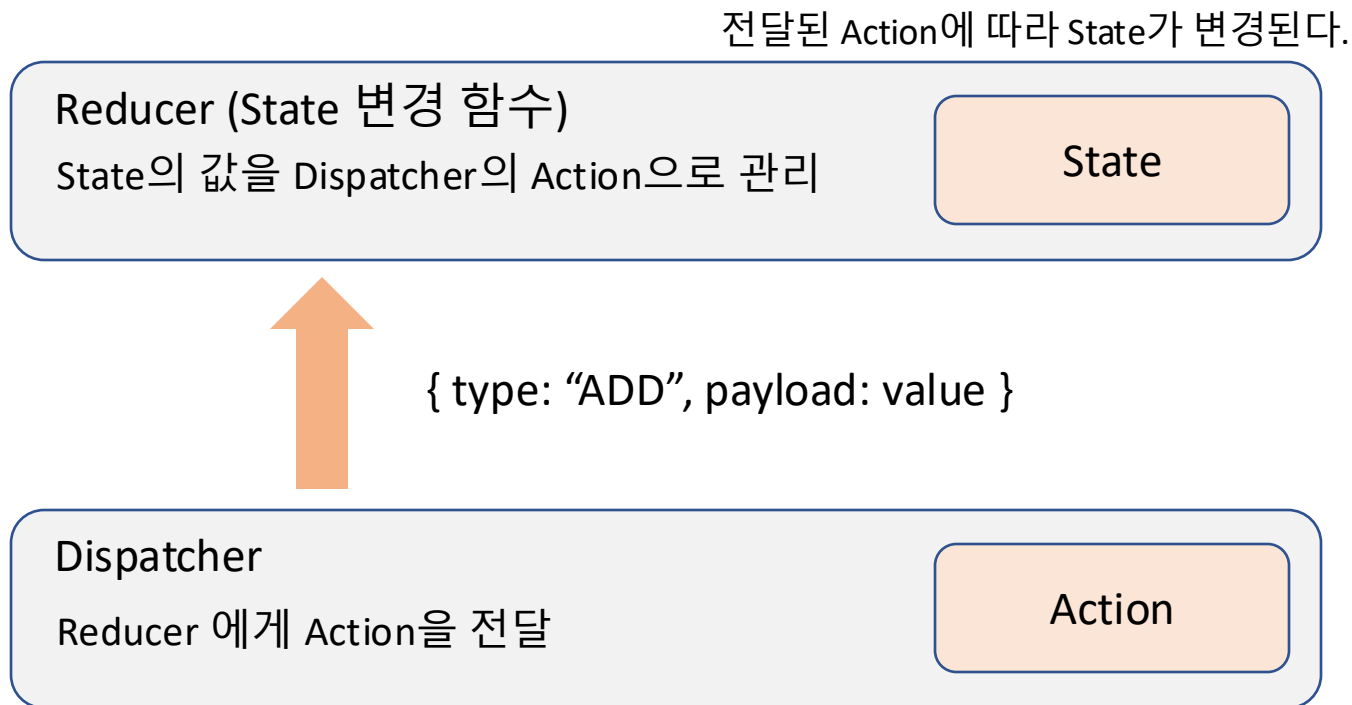
```
import { useRef, useState } from "react";
import TaskAppender from "../Components/TaskAppender";
import TaskList from "../Components/TaskList";
import Confirm from "../Components/modal/Confirm";
import Alert from "../Components/modal/Modal";

function App() {
  -- 생략 --
  return (
    <>
      <div className="wrapper">
        <header>React Todo</header>
        <TaskList>
          <TaskList.TaskHeader onCheckboxClick={doneAllTodoHandler} />
          {todoLists.map((item) => (
            <TaskList.TaskItem
              key={item.id}
              id={item.id}
              task={item.task}
              dueDate={item.dueDate}
              priority={item.priority}
              done={item.done}
              onCheckboxClick={doneAllTodoHandler}
            />
          )
        )}
        </TaskList>
      </div>
    </>
  );
}
```

13. useReducer

13. useReducer

- State를 변화시키는 복잡/다양한 함수들을 분리해서 관리하는 방법.



13. useReducer

- Todo Application의 State 변경 함수들을 Reducer로 분리
 - > /src/reducers/TaskReducers.jsx (1 / 2)

```
export const actionTypes = {
  done: "DONE",
  allDone: "ALL-DONE",
  add: "ADD",
};

export default function taskReducers(state, action) {
  const type = action.type;
  const payload = action.payload;

  if (type === actionTypes.add) {
    return [
      ...state,
      {
        id: "item" + (state.length + 1),
        task: payload.task,
        dueDate: payload.dueDate,
        priority: payload.priority,
        done: false,
      },
    ];
  }
}
```

13. useReducer

- Todo Application의 State 변경 함수들을 Reducer로 분리
 - > /src/reducers/TaskReducers.jsx (2 / 2)

```
} else if (type === actionTypes.done) {  
  return state.map((task) => {  
    if (task.id === payload.id) {  
      task.done = true;  
    }  
    return task;  
  });  
} else if (type === actionTypes.allDone) {  
  return state.map((task) => {  
    task.done = true;  
    return task;  
  });  
}  
return state;  
}
```

13. useReducer

- Todo Application의 State 변경 함수들을 Reducer로 분리
> /src/App.js (1 / 3)

```
import { useReducer, useRef, useState } from "react";
import TaskAppender from "../Components/TaskAppender";
import TaskList from "../Components/TaskList";
import Confirm from "../Components/modal/Confirm";
import Alert from "../Components/modal/Modal";
import taskReducers, { actionType } from "../reducers/TaskReducers";

function App() {
  const alertRef = useRef();
  const allDoneConfirmRef = useRef();
  const doneConfirmRef = useRef();

  const [allDoneConfirmMessage, setAllDoneConfirmMessage] = useState();
  const [alertMessage, setAlertMessage] = useState();

  const [todoLists, todoDispatcher] = useReducer(taskReducers, []);

  const addNewTodoHandler = (task, dueDate, priority) => {
```

13. useReducer

- Todo Application의 State 변경 함수들을 Reducer로 분리
> /src/App.js (2 / 3)

```
todoDispatcher({  
  type: actionTypes.add,  
  payload: { task, dueDate, priority },  
});  
};  
  
const doneTodoHandler = (event) => {  
  -- 생략 --  
};  
  
const doneTodoItemHandler = () => {  
  todoDispatcher({  
    type: actionTypes.done,  
    payload: { id: doneConfirmRef.todoId },  
  });  
  doneConfirmRef.current.close();  
};
```

13. useReducer

- Todo Application의 State 변경 함수들을 Reducer로 분리
> /src/App.js (3 / 3)

```
const doneAllTodoHandler = (event) => {  
  -- 생략 --  
};  
  
const allDoneOkHandler = () => {  
  todoDispatcher({ type: actionTypes.allDone, payload: {} });  
  allDoneConfirmRef.current.close();  
};  
  
return (  
  <>  
    -- 생략 --  
  </>  
);  
}  
  
export default App;
```

13. useReducer

- Context와 Reducer의 차이점
 - Context
 - Provider를 제공받는 컴포넌트에서 공유 가능.
 - Reducer
 - Provider가 없음
 - Reducer를 생성할 때 새로운 상태가 생성된다.
 - useState() 와 동일.
- 공유가능한 Reducer를 만드려면?
 - Context와 Reducer를 조합해야 한다.
 - Context의 State를 Reducer의 State로 사용
 - Context의 Action(Function)은 Dispatcher 호출

React Optimization

Optimization Technic

14. memo

15. useCallback

16. useMemo

14. memo

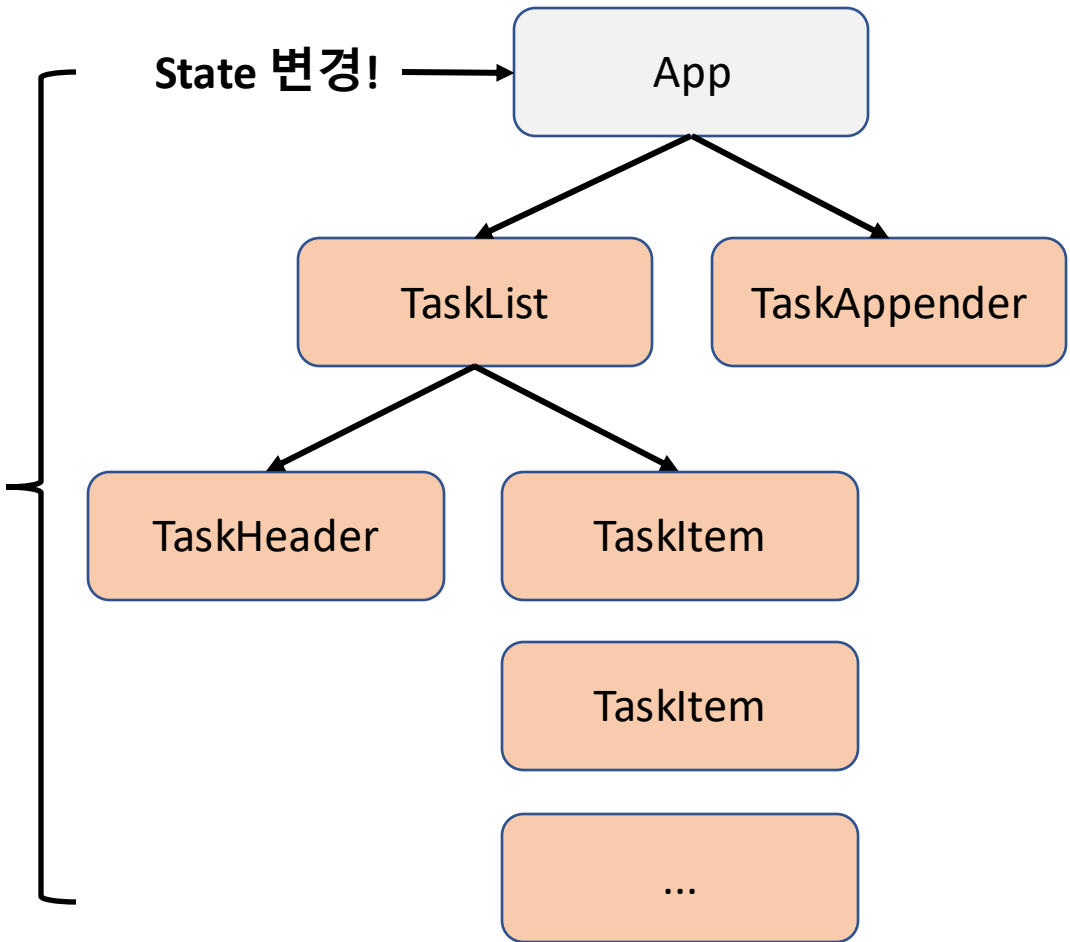
14. memo

- Memoization 의 Memo
- Memoization
 - 동일한 계산을 반복적으로 해야할 때, 이전에 계산한 값을 메모리에 저장하여, 중복적인 계산을 제거하는 방법
 - 전체적인 실행속도를 개선시킬 수 있다.
- memo
 - Component가 재실행되는 조건을 제어하는 함수
 - Props 또는 State가 변경되면, 해당 Component를 재실행하며, 해당 Component의 Child Component들도 모두 재실행 된다.
 - Props과 관련이 없는 Component는 재실행되지 않도록 막는 방법.
- Context를 사용하는 경우
 - Provider의 하위 컴포넌트에서는 memo가 동작하지 않는다.
 - 오직 Props에 대해서만 체크한다.

14. memo

**State가 변경된 컴포넌트가
재실행되며
하위 컴포넌트도 재실행된다!**

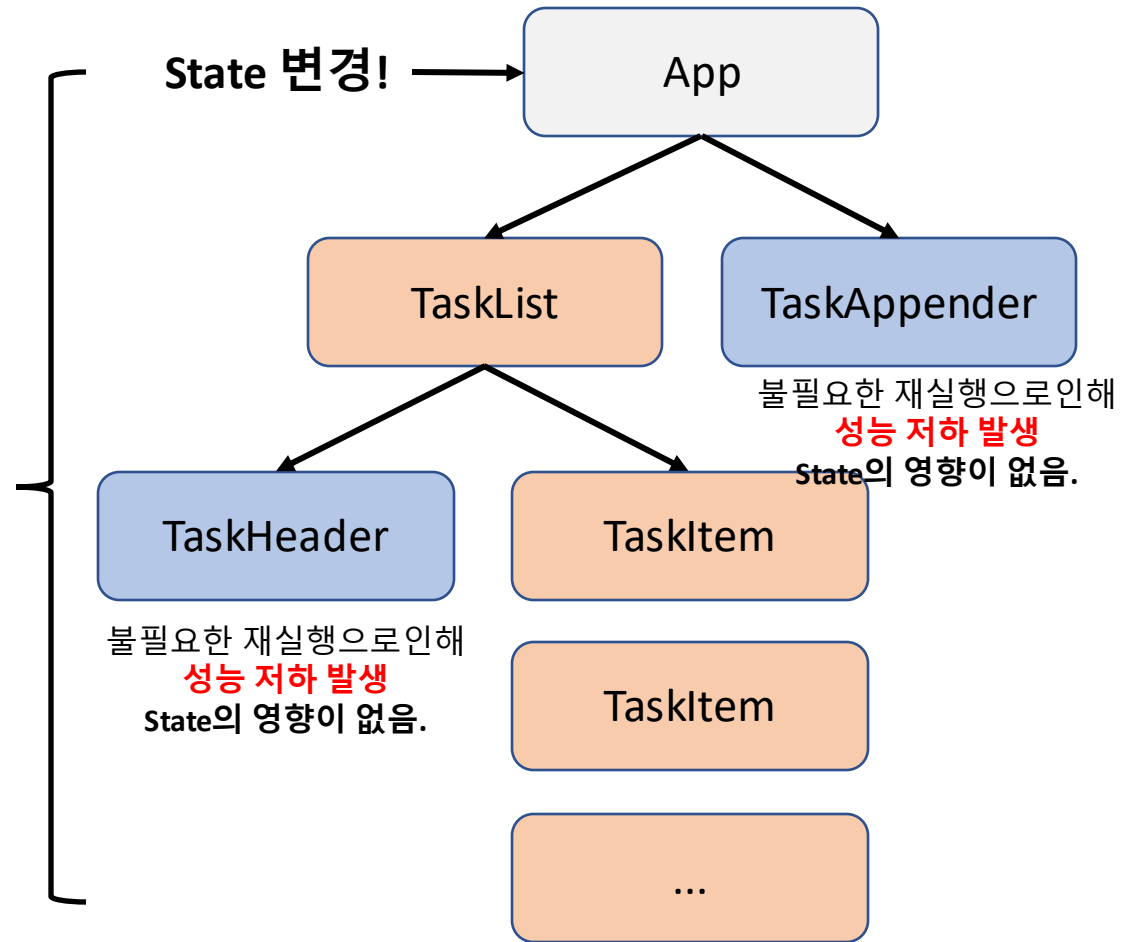
State기반으로 데이터를 표현하는
React에서는 당연한 프로세스



14. memo

**State가 변경된 컴포넌트가
재실행되며
하위 컴포넌트도 재실행된다!**

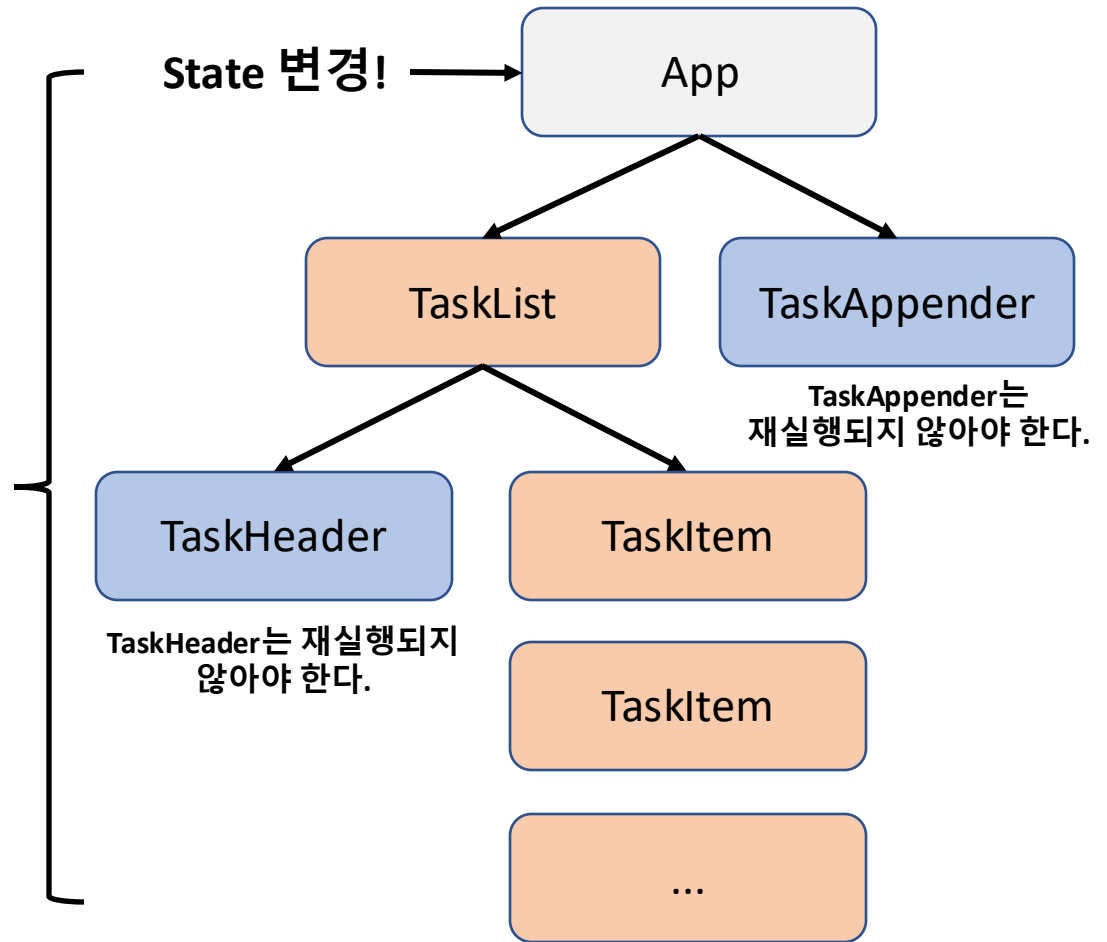
State기반으로 데이터를 표현하는
React에서는 당연한 프로세스



14. memo

**State가 변경된 컴포넌트가
재실행되며
하위 컴포넌트도 재실행된다!**

State기반으로 데이터를 표현하는
React에서는 당연한 프로세스



14. memo

- 컴포넌트마다 Console.log 기록

> /src/App.js

```
-- 생략 --  
function App() {  
  console.log("Call [App] Component");  
  console.log("Rendered [App] Component");  
  -- 생략 --  
}
```

> /src/Components/TaskAppender.js

```
-- 생략 --  
export default function TaskAppender({ onClick }) {  
  console.log("-- Call [TaskAppender] Component");  
  console.log("-- Rendered [TaskAppender] Component");  
  -- 생략 --  
}
```

14. memo

- 컴포넌트마다 Console.log 기록
> /src/Components/TaskList.js

```
-- 생략 --
export default function TaskList({ children }) {
  console.log("-- Call [TaskList] Component");
  console.log("-- Rendered [TaskList] Component");
  -- 생략 --
}

function TaskHeader({ onCheckboxClick }) {
  console.log("---- Call [TaskHeader] Component");
  console.log("---- Rendered [TaskHeader] Component");
  -- 생략 --
}

function TaskItem({ done, id, task, dueDate, priority, onCheckboxClick }) {
  console.log("---- Call [TaskItem] Component");
  console.log("---- Rendered [TaskItem] Component");
  -- 생략 --
}

TaskList.TaskHeader = TaskHeader;
TaskList.TaskItem = TaskItem;
```

14. memo

- 브라우저에서 Task 하나를 등록하면 TaskHeader, TaskAppender가 다시 실행된다.

Call [App] Component	App.js:9
Rendered [App] Component	App.js:10
Call [App] Component	App.js:9
Rendered [App] Component	App.js:10
-- Call [TaskList] Component	TaskList.jsx:6
-- Rendered [TaskList] Component	TaskList.jsx:7
-- Call [TaskList] Component	TaskList.jsx:6
-- Rendered [TaskList] Component	TaskList.jsx:7
---- Call [TaskHeader] Component	TaskList.jsx:18
---- Rendered [TaskHeader] Component	TaskList.jsx:19
---- Call [TaskHeader] Component	TaskList.jsx:18
---- Rendered [TaskHeader] Component	TaskList.jsx:19
-- Call [TaskAppender] Component	TaskAppender.jsx:5
-- Rendered [TaskAppender] Component	TaskAppender.jsx:6
-- Call [TaskAppender] Component	TaskAppender.jsx:5
-- Rendered [TaskAppender] Component	TaskAppender.jsx:6

14. memo

- 재실행이 필요 없는 TaskAppender, TaskHeader에 memo를 설정한다.

> /src/Components/TaskAppender.jsx

```
import { memo, useRef, useState } from "react";
import Alert from "../modal/Modal";

export default memo(function TaskAppender({ onClick }) {
  -- 생략 --
});
```

> /src/Components/TaskList.jsx > TaskHeader

```
import { createContext, memo, useContext } from "react";

-- 생략 --

TaskList.TaskHeader = memo(function TaskHeader({ onCheckboxClick }) {
  -- 생략 --
});

-- 생략 --
TaskList.TaskItem = TaskItem;
```

14. memo

- memo 는 Component에 전달되는 Props가 변경되었을 때만 컴포넌트를 다시 실행한다.

```
import { useRef, memo } from "react";  
export default memo(function AddTodo({ setTodo }) {  
  ...  
});
```

이전에 전달된 파라미터와 다른지 비교
이전에 전달된 파라미터와 같다면 컴포넌트를 다시 실행하지 않는다.

```
import { useRef, memo } from "react";  
export default memo(function AddTodo({ setTodo }) {  
  ...  
});
```

14. memo

- Memo 함수를 적용해도 TaskAppender와 TaskHeader는 재실행이 된다.
- 전달되는 Props의 형태에 따라 다르기 때문.
- TaskAppender와 TaskHeader의 Props는 function이다.
 - Javascript의 function은 Reference Type이다.
 - Props가 Reference Type일 경우 memo는 메모리 주소로 변경을 감지한다.
 - 이 경우 useCallback을 사용해야 한다.
 - 만약, Props가 Object 라면?
 - useMemo를 사용해야 한다.

14. memo

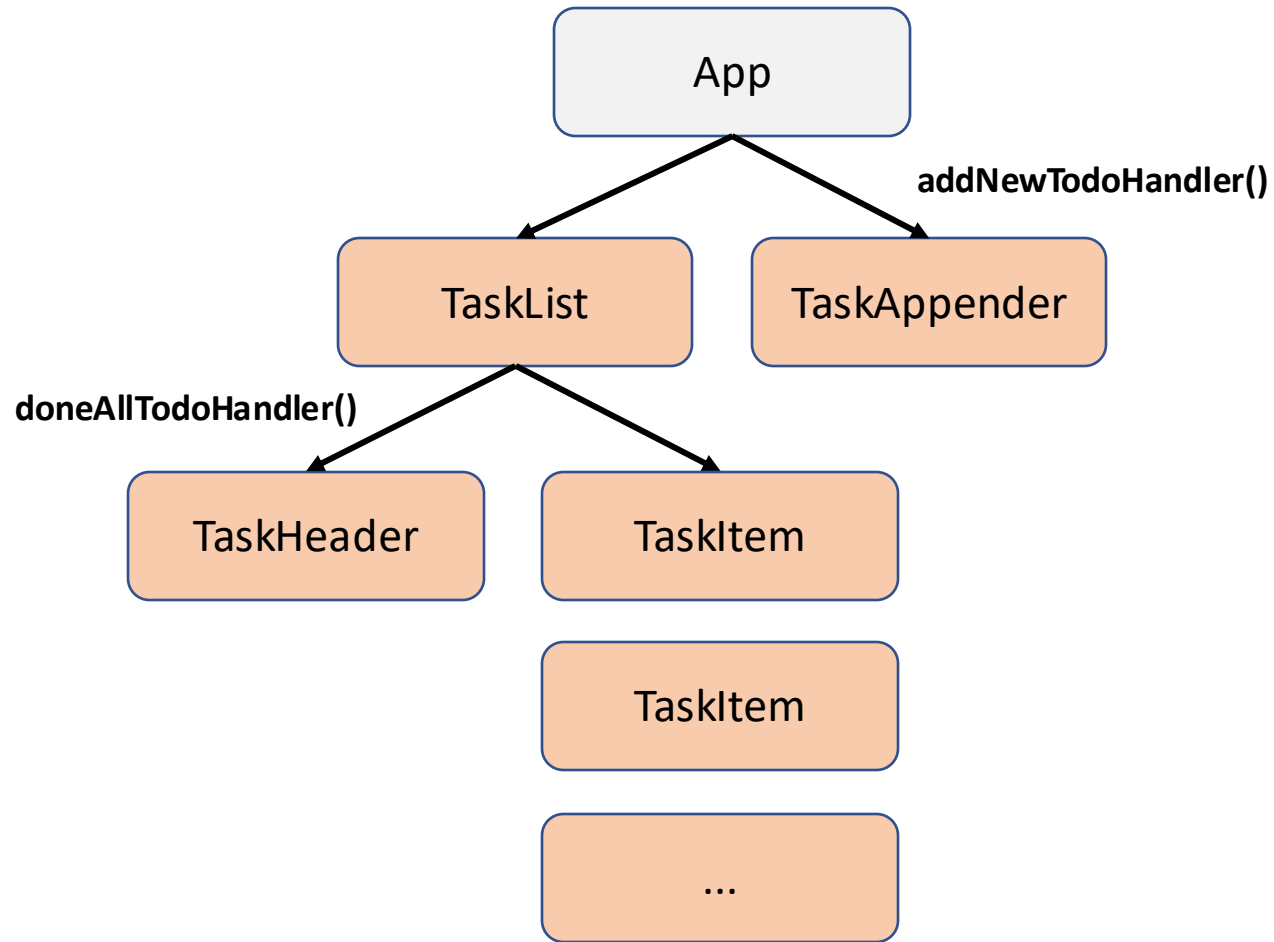
- memo 는 남발해서는 안됨!
 - Props를 비교하는 행위 자체가 매우 큰 비용의 연산.
 - 재실행이 자주 발생하거나, 많은 컴포넌트에서 발생할 경우
 - 오히려 성능이 저하된다.

15. useCallback

15. useCallback

- memo()와 유사한 동작방식을 가지고 있음.
- memo()는 컴포넌트의 재실행을 방지하는 방법.
- useCallback() Hook은 함수의 재 생성을 방지하는 방법
- props로 전달되는 함수의 재 생성을 방지
 - ➔ 하위 컴포넌트의 재 실행을 방지.

15. useCallback



15. useCallback

- Javascript에서 function 은 객체(Reference)로 취급된다.



App

```
const fn = () => {}  
<TaskAppender fn={fn} />
```



memo(TaskAppender({ fn })))

재실행 전 과
재실행 후 의
fn함수는 다른 함수!

컴포넌트 함수가
재실행되면서
컴포넌트 내부의 함수도
새롭게 생성되기 때문.

State 변경 후 재실행



App

```
const fn = () => {}  
<TaskAppender fn={fn} />
```



memo(TaskAppender({ fn })))

15. useCallback

- 하위 컴포넌트로 전달되는 함수가 변경되지 않도록 보장하는
- useCallback Hook

> /src/App.js (1 / 2)

```
import { useCallback, useReducer, useRef, useState } from "react";
import TaskAppender from "../Components/TaskAppender";
import TaskList from "../Components/TaskList";
import Confirm from "../Components/modal/Confirm";
import Alert from "../Components/modal/Modal";
import taskReducers, { actionTypes } from "../reducers/TaskReducers";

function App() {
  -- 생략 --

  const addNewTodoHandler = useCallback((task, dueDate, priority) => {
    -- 생략 --
  }, []);

  const doneTodoHandler = (event) => {
    -- 생략 --
  };
}
```

15. useCallback

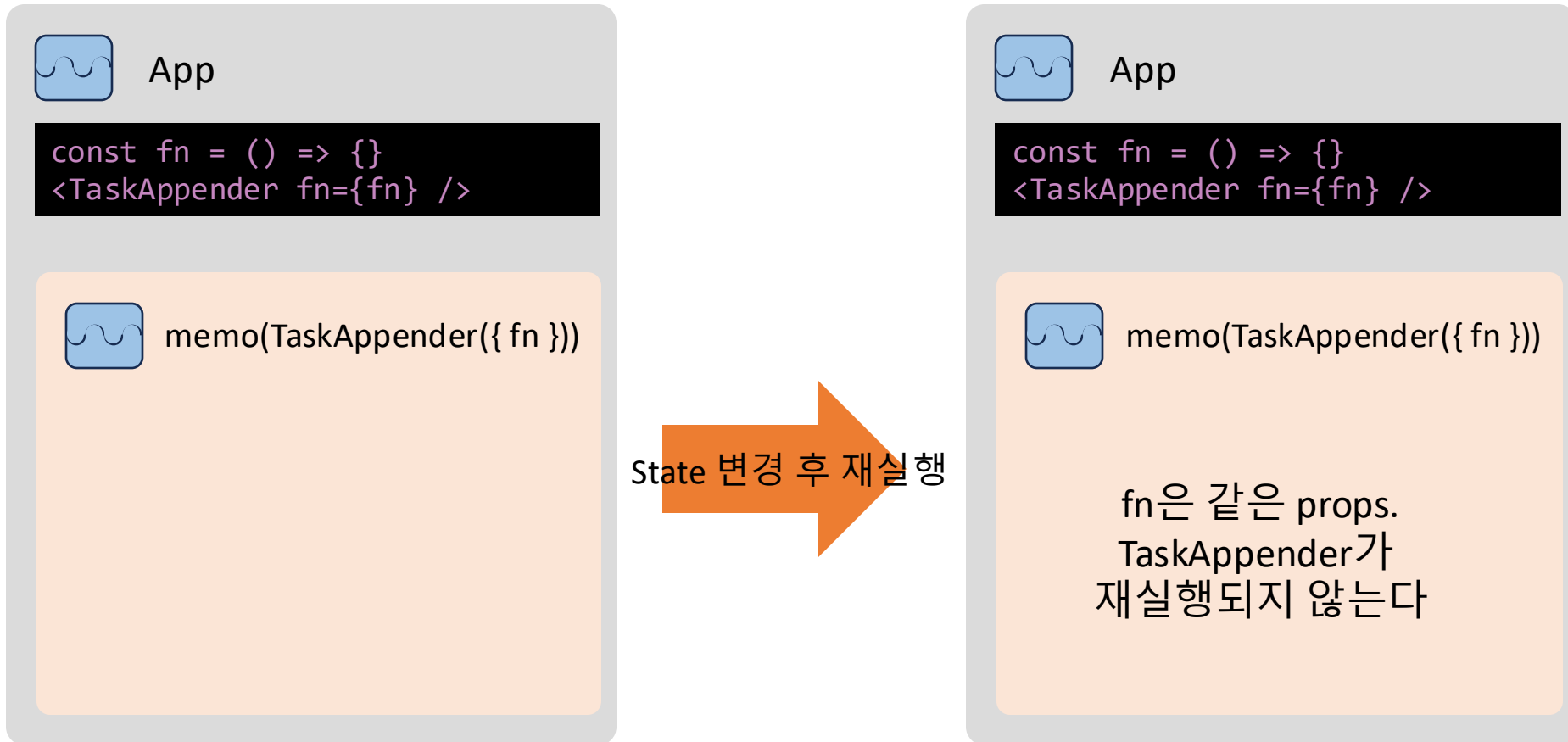
- 하위 컴포넌트로 전달되는 함수가 변경되지 않도록 보장하는
- useCallback Hook

> /src/App.js (2 / 2)

```
const doneTodoItemHandler = () => {  
  -- 생략 --  
};  
  
const doneAllTodoHandler = useCallback((event) => {  
  -- 생략 --  
}, [todoLists]);  
  
const allDoneOkHandler = () => {  
  -- 생략 --  
};  
  
return (  
  -- 생략 --  
);  
}  
  
export default App;
```

15. useCallback

- useCallback으로 정의된 함수는 컴포넌트가 재실행되더라도
- 새로운 함수로 재생성되지 않는다.

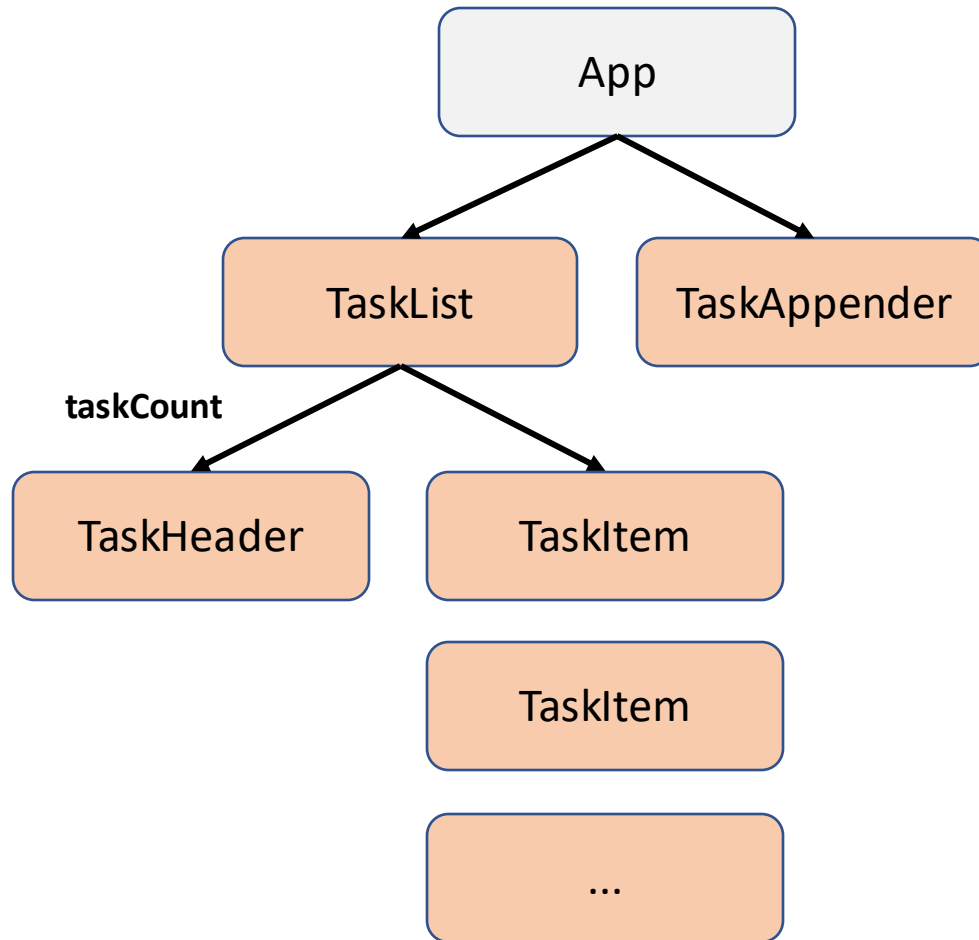


16. useMemo

16. useMemo

- useCallback() Hook과 매우 유사함.
- useCallback() Hook은 함수의 재 생성을 방지하는 방법
- useMemo() Hook은 객체의 재 생성을 방지하는 방법
- props로 전달되는 객체의 재 생성을 방지
 - ➔ 하위 컴포넌트의 재 실행을 방지.

16. useMemo



16. useMemo

- TaskHeader 컴포넌트로 taskCount 객체 전달하기

> /src/App.js (1 / 2)

-- 생략 --

```
function App() {
```

-- 생략 --

```
  const [todoLists, todoDispatcher] = useReducer(taskReducers, []);
```

```
  const taskCount = {  
    done: todoLists.filter((item) => item.done).length,  
    process: todoLists.filter((item) => !item.done).length,  
  };
```

-- 생략 --

```
  return (
```

```
    <>
```

```
    <div className="wrapper">
```

```
      <header>React Todo</header>
```

```
      <TaskList>
```

```
        <TaskList.TaskHeader
```

16. useMemo

- TaskHeader 컴포넌트로 taskCount 객체 전달하기

> /src/App.js (2 / 2)

```
    taskCount={taskCount}
    onCheckboxClick={doneAllTodoHandler}
  />
  -- 생략 --
</TaskList>
<TaskAppender onClick={addNewTodoHandler} />
</div>
-- 생략 --
</>
);
}
```

export default App;

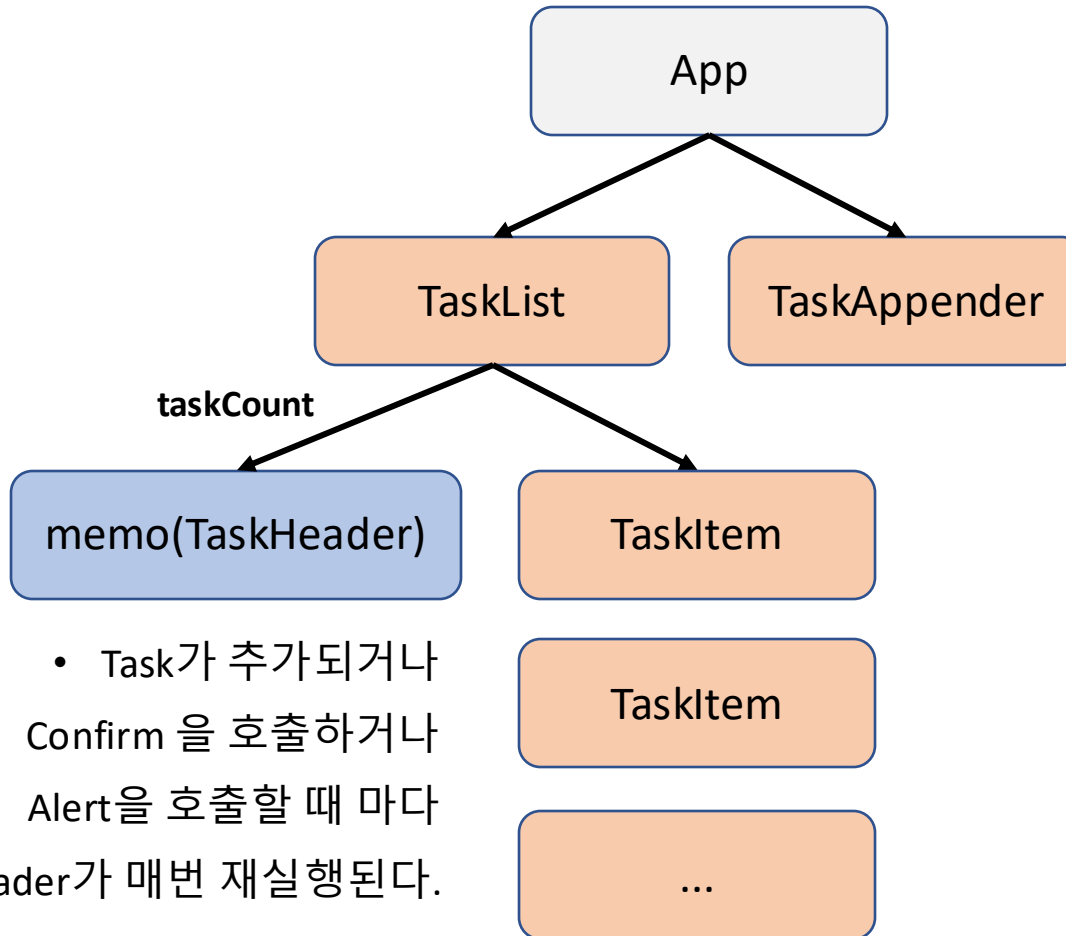
16. useMemo

- Todo, AddTodo 컴포넌트로 Style 객체를 전달.

> /src/Components/TaskList.js > TaskHeader

```
-- 생략 --
TaskList.TaskHeader = memo(function TaskHeader({taskCount, onCheckboxClick}) {
  -- 생략 --
  return (
    <>
      <li className="tasks-counter">
        <div>진행중: {taskCount.process}</div>
        <div>완료: {taskCount.done}</div>
      </li>
      <li className="tasks-header">
        -- 생략 --
      </li>
    </>
  );
});
-- 생략 --
```

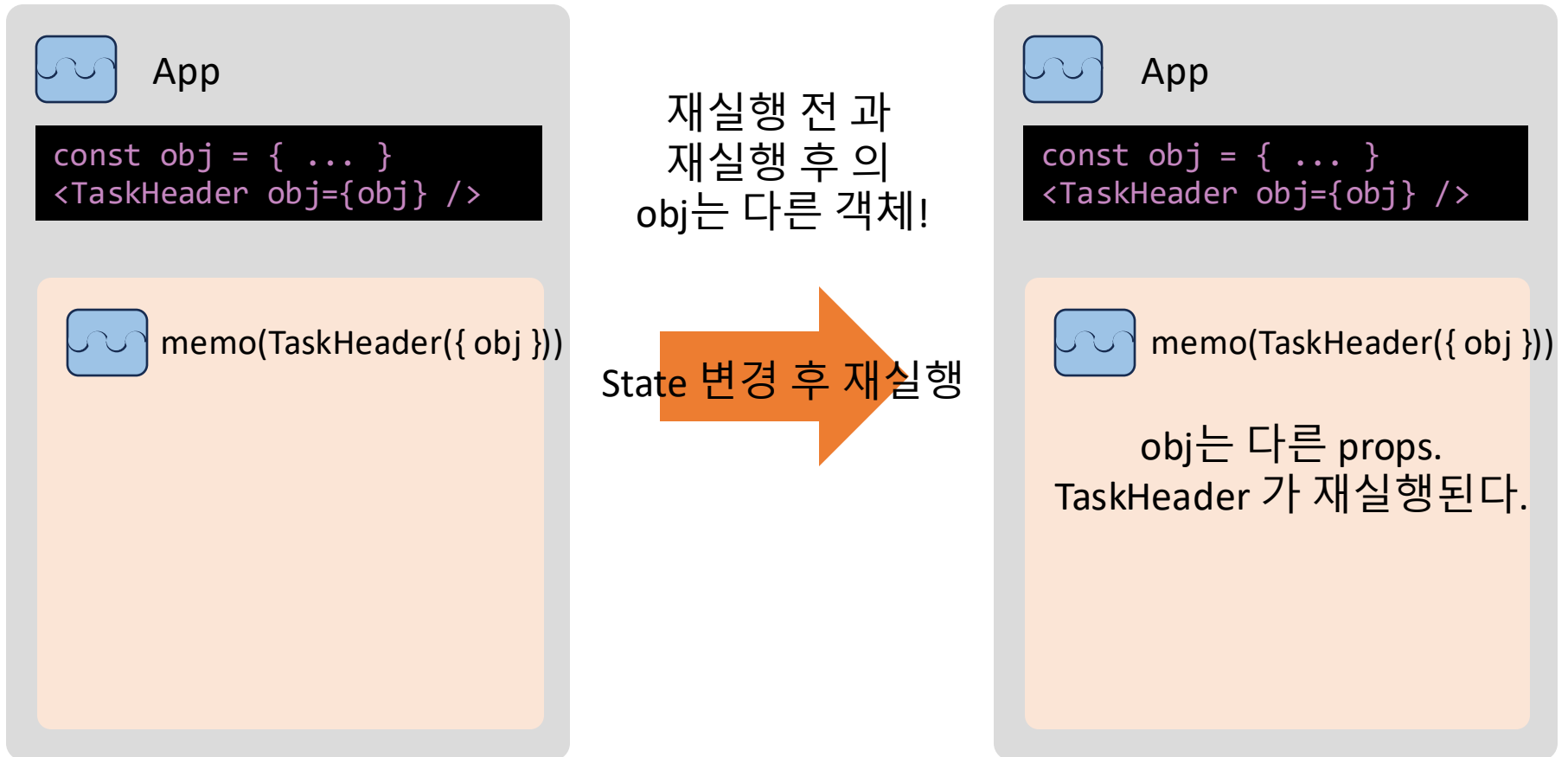
16. useMemo



- Task가 추가되거나
- Confirm 을 호출하거나
- Alert을 호출할 때 마다
- TaskHeader가 매번 재실행된다.

16. useMemo

- Javascript에서 객체리터럴은 객체(Reference)로 취급된다.



16. useMemo

- Todo, AddTodo 컴포넌트로 전달되는 객체를 useMemo()로 처리

> /src/components/TodoApp.js

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

function App() {
  -- 생략 --
  const [todoLists, todoDispatcher] = useReducer(taskReducers, []);

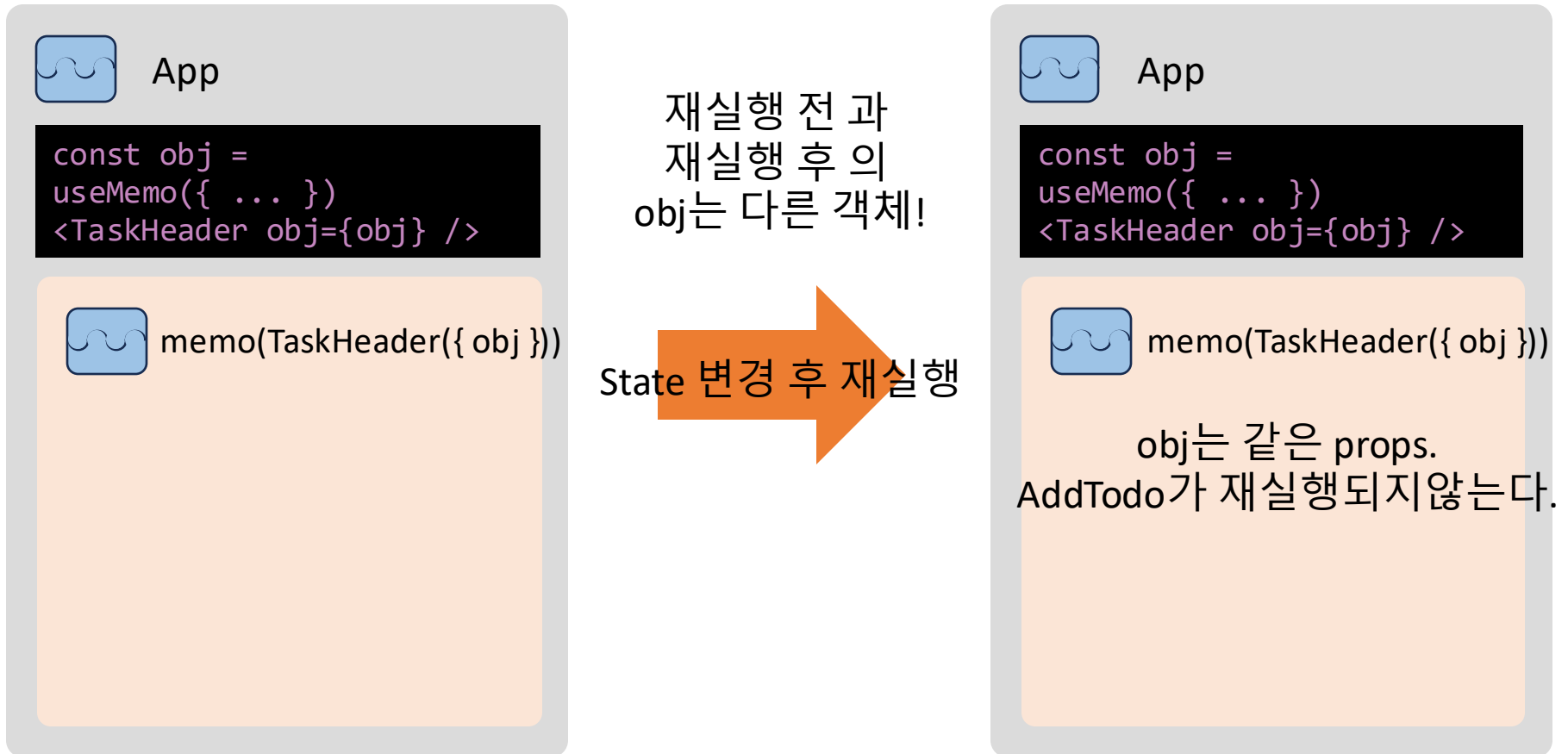
  const taskCount = useMemo(() => {
    return {
      done: todoLists.filter((item) => item.done).length,
      process: todoLists.filter((item) => !item.done).length,
    };
  }, [todoLists]);

  -- 생략 --
}

export default App;
```

16. useMemo

- useMemo로 정의된 객체리터럴은 컴포넌트가 재실행되더라도
- 새로운 객체리터럴로 재생성되지 않는다.



RESTful API Service

Fetch api

17. fetch

18. side effect

19. fetch를 이용해 Todo App 완성해보기

17. fetch

17. fetch

- Javascript 환경에서 사용가능한 비동기 통신 방법
 1. XMLHttpRequest (전통적 방식)
 2. Promise (ECMA6)
 3. **window.fetch (Promise 기반의 Browser API)**
 4. 기타 Libraries

17. fetch

- fetch API
 - Promise based Javascript Built-in Function (React 전용 함수가 아님)
 - `http://localhost:8888/api/v1/task` (GET) 으로 요청하는 방법

```
fetch("http://localhost:8888/api/v1/task", {
  method: "GET",
})
.then((response) => {
  console.log("Response: ", response);
  return response.json();
})
.then((json) => {
  console.log("Json: ", json);
})
.catch((rejectReason) => {
  console.log("Reject: ", rejectReason);
});
```

17. fetch

- fetch : REST API 호출.
- then : REST API의 결과가 정상적으로 도착했을 때 실행할 Callback
- catch : REST API가 비정상적으로 종료되었을 때 실행할 Callback

```
fetch("http://localhost:8888/api/v1/task", {  
  method: "GET",  
})  
  .then((response) => {  
    console.log("Response: ", response);  
    return response.json();  
  })  
  .then((json) => {  
    console.log("Json: ", json);  
  })  
  .catch((rejectReason) => {  
    console.log("Reject: ", rejectReason);  
  });
```

17. fetch

- Promise의 응답 데이터를 전송 받기 위해 .then().then()... 을 사용
- Promise를 간편하게 사용하기 위한 async / await
 - await : 비동기 함수를 동기 통신처럼 응답이 올 때까지 대기한다.
 - async 가 적용된 함수 내에서만 사용 가능.
 - async : 비동기 함수임을 나타냄

```
const fetchCall = async () => {  
  const response =  
    await fetch("http://localhost:8888/api/v1/task", {  
      method: "GET",  
    });  
  console.log("Response: ", response);  
  
  const json = await response.json();  
  console.log("JSON: ", json);  
};  
fetchCall();
```

18. Side effects

18. Side effects

- Side Effect = 어떤 함수가 실행됨으로써 화면의 변화가 일어나는 것
- React Component는 Props와 State가 변경될 경우, Component를 재 실행해 화면을 다시 그린다.
 - ➔ Side Effect
- 비동기 통신 또는 이벤트에 의해 Props혹은 State가 변경될 경우 Component를 무한히 재실행하는 부작용이 생길 수 있다.

18. Side effects

- REST API를 이용해 응답을 받아와 State에 할당해보기

> /src/reducers/TaskReducers.jsx

```
export const actionTypes = {
  done: "DONE",
  allDone: "ALL-DONE",
  add: "ADD",
  init: "INIT",
};

export default function taskReducers(state, action) {
  const type = action.type;
  const payload = action.payload;

  if (type === actionTypes.init) {
    return [...payload];
  } else if (type === actionTypes.add) {
    -- 생략 --
  } else if (type === actionTypes.done) {
    -- 생략 --
  } else if (type === actionTypes.allDone) {
    -- 생략 --
  }
  return state;
}
```

18. Side effects

- REST API를 이용해 응답을 받아와 State에 할당해보기

> /src/App.js (1 / 3)

-- 생략 --

```
function App() {  
  console.log("Call [App] Component");  
  console.log("Rendered [App] Component");  
}
```

-- 생략 --

```
const [todoLists, todoDispatcher] = useReducer(taskReducers, []);
```

```
const [isLoading, setIsLoading] = useState(true);
```

```
const taskFetch = async () => {  
  setIsLoading(true);
```

```
  const response = await fetch("http://localhost:8888/api/v1/task");  
  const json = await response.json();  
  setIsLoading(false);  
  todoDispatcher({ type: actionTypes.init, payload: json.body });  
};
```

```
taskFetch();
```

18. Side effects

- REST API를 이용해 응답을 받아와 State에 할당해보기

> /src/App.js (2 / 3)

```
const taskCount = useMemo(() => {  
  -- 생략 --  
}, [todoLists]);  
  
-- 생략 --  
  
return (  
  <>  
    <div className="wrapper">  
      <header>React Todo</header>  
      <TaskList>  
        <TaskList.TaskHeader  
          taskCount={taskCount}  
          onCheckboxClick={doneAllTodoHandler}  
        />  
        {isLoading && <div>Loading...</div>}  
        {!isLoading &&  
          todoLists.map((item) => (  
            <TaskList.TaskItem
```


18. Side effects

- REST API를 이용해 응답을 받아와 State에 할당해보기

> /src/App.js (3 / 3)

```
    -- 생략 --
  />
  )}}
</TaskList>
<TaskAppender onClick={addNewTodoHandler} />
</div>
-- 생략 --
</>
);
}

export default App;
```

18. Side effects

- REST API를 이용해 응답을 받아와 State에 할당해보기

✖ ▶ Uncaught Error: Too many re-renders. React [react-dom-client.development.js:5613](#) limits the number of renders to prevent an infinite loop.

- at renderWithHooksAgain ([react-dom-client.development.js:5613:1](#))
- at renderWithHooks ([react-dom-client.development.js:5531:1](#))
- at updateFunctionComponent ([react-dom-client.development.js:8897:1](#))
- at beginWork ([react-dom-client.development.js:10522:1](#))
- at runWithFiberInDEV ([react-dom-client.development.js:1518:1](#))
- at performUnitOfWork ([react-dom-client.development.js:15130:1](#))
- at workLoopSync ([react-dom-client.development.js:14956:1](#))
- at renderRootSync ([react-dom-client.development.js:14936:1](#))
- at performWorkOnRoot ([react-dom-client.development.js:14462:1](#))
- at performWorkOnRootViaSchedulerTask ([react-dom-client.development.js:16216:1](#))

⚠ ▶ An error occurred in the <App> component. [index.js:9](#)

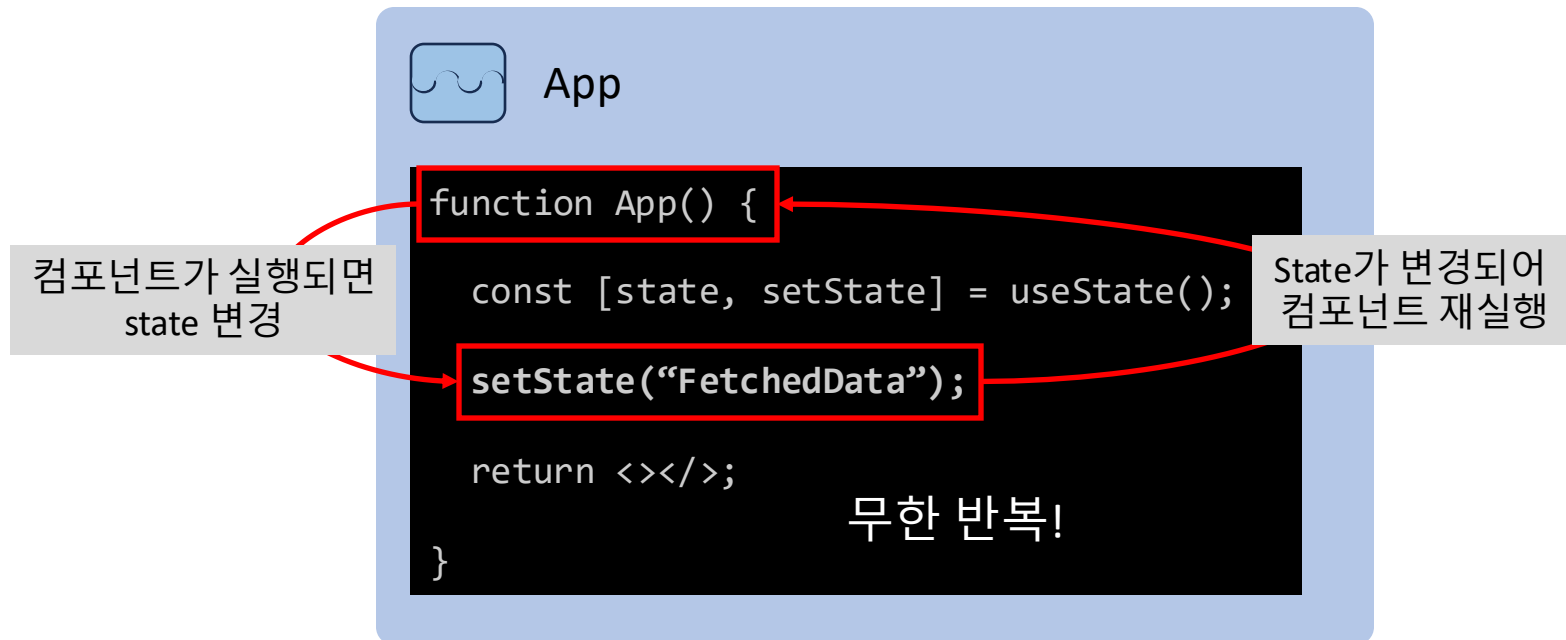
Consider adding an error boundary to your tree to customize error handling behavior.

Visit <https://react.dev/link/error-boundaries> to learn more about error boundaries.

✖ ▶ Can't perform a React state update on a component that hasn't mounted [App.js:27](#) yet. This indicates that you have a side-effect in your render function that asynchronously later calls tries to update the component. Move this work to `useEffect` instead.

18. Side effects

- 에러가 발생하는 원인
 - State / Props가 변경되면 해당 컴포넌트는 항상 재실행 된다.



18. Side effects

- 해결하는 방법?
 - 특별한 상황일 때 만 state가 변경되도록 한다.

```
useEffect(실행할 함수, 의존 배열);
```

> Component가 처음 실행될 때 단 한번만 함수가 실행된다.

```
useEffect(() => {  
  console.log("Callback")  
}, []);
```

> aState가 변경될 경우 함수가 실행된다. > bProps가 변경될 경우 함수가 실행된다.

```
useEffect(() => {  
  console.log("Callback")  
}, [aState]);
```

```
useEffect(() => {  
  console.log("Callback")  
}, [bProps]);
```

> aState또는 bProps가 변경될 경우 함수가 실행된다.

```
useEffect(() => {  
  console.log("Callback")  
}, [aState, bProps]);
```

18. Side effects

- 컴포넌트가 처음 실행되면 fetch해, state를 변경한다.

> /src/App.js

```
import {
  useCallback,
  useEffect,
  useMemo,
  useReducer,
  useRef,
  useState,
} from "react";
-- 생략 --

function App() {
  -- 생략 --
  useEffect(() => {
    taskFetch();
  }, []);

  -- 생략 --
}
```

18. Side effects

- 컴포넌트가 처음 실행될 때 한번만 실행.
- 무한반복이 발생하지 않는다.

19. Fetch를 이용해 Todo App 완성해보기

19. Fetch를 이용해 Todo App 완성해보기

- API Server 준비
 - <https://github.com/CodeMakers-KR/react-todo-api-server/releases>
- (GET) `http://localhost:8888/api/v1/task`
 - Task 목록 불러오기
- (POST) `http://localhost:8888/api/v1/task`
 - Task 등록하기
- (PUT) `http://localhost:8888/api/v1/task/{taskId}`
 - Task 완료 처리하기
- (PUT) `http://localhost:8888/api/v1/task`
 - 모든 Task 완료 처리하기

19. Fetch를 이용해 Todo App 완성해보기

- (GET) `http://localhost:8888/api/v1/task`
 - Task 목록 불러오기
 - Response

```
{
  "status": 200,
  "statusMessage": "OK",
  "pages": 0,
  "next": false,
  "errors": null,
  "count": 6,
  "body": [
    {
      "taskId": "task_0",
      "task": "test task",
      "dueDate": "2025-12-11",
      "priority": "1",
      "done": true
    }, ...
  ]
}
```

19. Fetch를 이용해 Todo App 완성해보기

- (POST) `http://localhost:8888/api/v1/task`

- Task 등록하기

- Request

```
{  
  "task": "test task",  
  "dueDate": "2025-12-11",  
  "priority": "1",  
  "isDone": false  
}
```

- Response

```
{  
  "status": 201,  
  "statusMessage": "Created",  
  "pages": 0,  
  "next": false,  
  "errors": null,  
  "count": 0,  
  "body": {  
    "taskId": "task_6",  
    "task": "test task",  
    "dueDate": "2025-12-11",  
    "priority": "1",  
    "done": false  
  }  
}
```

19. Fetch를 이용해 Todo App 완성해보기

- Task 등록.

> /src/reducers/TaskReducers.jsx (1 / 2)

-- 생략 --

```
export default function taskReducers(state, action) {  
  const type = action.type;  
  const payload = action.payload;  
  
  if (type === actionTypes.init) {  
    return [...payload];  
  } else if (type === actionTypes.add) {  
    return [  
      ...state,  
      {  
        id: payload.taskId,  
        task: payload.task,  
        dueDate: payload.dueDate,  
        priority: payload.priority,  
        done: false,  
      },  
    ];  
  } else if (type === actionTypes.done) {
```

19. Fetch를 이용해 Todo App 완성해보기

- Task 등록.

> /src/reducers/TaskReducers.jsx (2 / 2)

```
-- 생략 --  
} else if (type === actionTypes.allDone) {  
  -- 생략 --  
}  
return state;  
}
```

19. Fetch를 이용해 Todo App 완성해보기

- Task 등록.

> /src/App.js (1 / 2)

-- 생략 --

```
function App() {
```

-- 생략 --

```
const addNewTodoHandler = useCallback((task, dueDate, priority) => {
```

```
  const addFetch = async (fnCallback) => {
```

```
    const response = await fetch("http://localhost:8888/api/v1/task", {
```

```
      method: "post",
```

```
      headers: {
```

```
        "Content-Type": "application/json",
```

```
      },
```

```
      body: JSON.stringify({
```

```
        task,
```

```
        dueDate,
```

```
        priority,
```

```
        isDone: false,
```

```
      }},
```

```
    });
```

19. Fetch를 이용해 Todo App 완성해보기

- Task 등록.

> /src/App.js (2 / 2)

```
const json = await response.json();
if (json.status === 201) {
  fnCallback(json.body.taskId);
}
};

addFetch((taskId) => {
  todoDispatcher({
    type: actionTypes.add,
    payload: { taskId, task, dueDate, priority },
  });
});
}, []);

-- 생략 --

}

export default App;
```

19. Fetch를 이용해 Todo App 완성해보기

- (PUT) `http://localhost:8888/api/v1/task/{taskId}`
 - Task 완료 처리하기

- Response

```
{  
  "status": 200,  
  "statusMessage": "OK",  
  "pages": 0,  
  "next": false,  
  "errors": null,  
  "count": 0,  
  "body": "{taskId}"  
}
```

19. Fetch를 이용해 Todo App 완성해보기

- Task 완료.

> /src/App.js (1 / 2)

```
-- 생략 --
function App() {
  -- 생략 --
  const doneTodoItemHandler = () => {
    const doneFetch = async (fnCallback) => {
      const response = await fetch(
        `http://localhost:8888/api/v1/task/${doneConfirmRef.todoId}`,
        {
          method: "put",
          headers: {
            "Content-Type": "application/json",
          },
        },
      );

      const json = await response.json();
      if (json.status === 200) {
        fnCallback(json.body.taskId);
      }
    };
  };
}
```


19. Fetch를 이용해 Todo App 완성해보기

- Task 완료.

> /src/App.js (2 / 2)

```
doneFetch(() => {  
  todoDispatcher({  
    type: actionTypes.done,  
    payload: { id: doneConfirmRef.todoId },  
  });  
  doneConfirmRef.current.close();  
});  
  
-- 생략 --  
  
}  
  
export default App;
```

19. Fetch를 이용해 Todo App 완성해보기

- (PUT) <http://localhost:8888/api/v1/task>

- 모든 Task 완료 처리하기

- Response

```
{  
  "status": 200,  
  "statusMessage": "OK",  
  "pages": 0,  
  "next": false,  
  "errors": null,  
  "count": 0,  
  "body": null  
}
```

19. Fetch를 이용해 Todo App 완성해보기

- 모든 Task 완료.

> /src/App.js (1 / 2)

```
-- 생략 --
function App() {
  -- 생략 --
  const allDoneOkHandler = () => {
    const allDoneFetch = async (fnCallback) => {
      const response = await fetch("http://localhost:8888/api/v1/task", {
        method: "put",
        headers: {
          "Content-Type": "application/json",
        },
      });
    };

    const json = await response.json();
    if (json.status === 200) {
      fnCallback();
    }
  };

  allDoneFetch(() => {
    todoDispatcher({ type: actionTypes.allDone, payload: {} });
  });
}
```

19. Fetch를 이용해 Todo App 완성해보기

- 모든 Task 완료.

> /src/App.js (2 / 2)

```
    allDoneConfirmRef.current.close();  
  });  
  -- 생략 --  
}
```

Custom React Hooks

20. Custom react hook

20. Custom react hook

20. Custom react hook

- React에서 제공하는 Hook 외에 개발자가 직접 만드는 Hook
- React에서 제공하는 Hook
 - useState
 - useEffect
 - useReducer
 - useCallback
 - useMemo
 - etc.

20. Custom react hook

- 함수의 이름을 use로 시작하면 Custom Hook으로 취급한다.
- Custom Hook 이 필요한 이유.
 - Component 내부에서 여러 Hook 의 결합으로 사용하는 것이 복잡하고 읽기 쉽지 않음.
 - 하나의 Hook내부에서 여러 Hook 들을 결합해 사용하기 위해.
- Custom Hook은 Component 내부에서만 사용 가능
- 다른 Hook 혹은 함수 내부에서는 사용할 수 없다.
 - React Hook도 동일함.

20. Custom react hook

- http/taskHttp.js 파일을 만들어 Fetch 코드를 모두 옮긴다.
> /src/http/http.js (1 / 4)

```
export async function loadTasks() {
  const response = await fetch("http://localhost:8888/api/v1/task");

  const json = await response.json();

  if (json.status === 200) {
    return json;
  }

  throw new Error(response.statusMessage);
}

export async function addTask({ task, dueDate, priority }) {
  const response = await fetch("http://localhost:8888/api/v1/task", {
    method: "post",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      task,
```

20. Custom react hook

- http/taskHttp.js 파일을 만들어 Fetch 코드를 모두 옮긴다.
> /src/http/http.js (2 / 4)

```
    dueDate,  
    priority,  
    isDone: false,  
  }},  
});  
  
const json = await response.json();  
if (json.status === 201) {  
  return json;  
}  
  
throw new Error(response.statusMessage);  
}  
  
export async function doneTask(taskId) {  
  const response = await fetch(`http://localhost:8888/api/v1/task/${taskId}`, {  
    method: "put",  
    headers: {  
      "Content-Type": "application/json",  
    },  
  },
```

20. Custom react hook

- http/taskHttp.js 파일을 만들어 Fetch 코드를 모두 옮긴다.
> /src/http/http.js (3 / 4)

```
});

const json = await response.json();
if (json.status === 200) {
  return json;
}

throw new Error(json.statusMessage);
}

export async function allDoneTasks() {
  const response = await fetch("http://localhost:8888/api/v1/task", {
    method: "put",
    headers: {
      "Content-Type": "application/json",
    },
  });
}

const json = await response.json();
if (json.status === 200) {
```

20. Custom react hook

- http/taskHttp.js 파일을 만들어 Fetch 코드를 모두 옮긴다.
> /src/http/http.js (4 / 4)

```
    return json;
  }

  throw new Error(json.statusMessage);
}
```

20. Custom react hook

- http function 이용해보기. (Custom Hook 사용 전)
> /src/App.js (1 / 5)

```
-- 생략 --
import { addTask, allDoneTasks, doneTask, loadTasks } from "../http/taskHttp";

function App() {
  -- 생략 --
  const taskFetch = async () => {
    setIsLoading(true);

    try {
      const json = await loadTasks();
      todoDispatcher({ type: actionTypes.init, payload: json.body });
    } catch (e) {
      alert(e.message || "데이터 조회 실패");
    } finally {
      setIsLoading(false);
    }
  };

  useEffect(() => {
    taskFetch();
  });
}
```

20. Custom react hook

- http function 이용해보기. (Custom Hook 사용 전)
> /src/App.js (2 / 5)

```
}, []);
```

```
-- 생략 --
```

```
const addNewTodoHandler = useCallback((task, dueDate, priority) => {  
  const addFetch = async (fnCallback) => {  
    try {  
      const json = await addTask({ task, dueDate, priority });  
      fnCallback(json.body.taskId);  
    } catch (e) {  
      alert(e.message || "task 등록 실패");  
    }  
  };  
};
```

```
addFetch((taskId) => {  
  todoDispatcher({  
    type: actionTypes.add,  
    payload: { taskId, task, dueDate, priority },  
  });  
});
```

20. Custom react hook

- http function 이용해보기. (Custom Hook 사용 전)
> /src/App.js (3 / 5)

```
}, []);

-- 생략 --

const doneTodoItemHandler = () => {
  const doneFetch = async (fnCallback) => {
    try {
      const json = await doneTask(doneConfirmRef.todoId);
      fnCallback(json.body.taskId);
    } catch (e) {
      alert(e.message || "task 완료 실패");
    }
  };

  doneFetch(() => {
    todoDispatcher({
      type: actionTypes.done,
      payload: { id: doneConfirmRef.todoId },
    });
    doneConfirmRef.current.close();
  });
}
```

20. Custom react hook

- http function 이용해보기. (Custom Hook 사용 전)
> /src/App.js (4 / 5)

```
});  
};  
  
-- 생략 --  
  
const allDoneOkHandler = () => {  
  const allDoneFetch = async (fnCallback) => {  
    try {  
      await allDoneTasks();  
      fnCallback();  
    } catch (e) {  
      alert(e.message || "task 완료 실패");  
    }  
  };  
};  
  
allDoneFetch(() => {  
  todoDispatcher({ type: actionTypes.allDone, payload: {} });  
  allDoneConfirmRef.current.close();  
});  
};
```


20. Custom react hook

- http function 이용해보기. (Custom Hook 사용 전)
> /src/App.js (5 / 5)

```
-- 생략 --  
}  
  
export default App;
```

20. Custom react hook

- Custom Hook 으로 http function 이용해보기.
> /src/hooks/useFetch.js (1 / 2)

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

export function useFetch(initialValue, fnFetch) {
  const [fetchData, setFetchData] = useState(initialValue);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState();

  useEffect(() => {
    setIsLoading(true);
    setError(undefined);
    const fetchCall = async () => {
      try {
        const json = await fnFetch();
        setFetchData(json);
      } catch (e) {
        setError(e.message || "데이터 처리 실패");
      } finally {
        setIsLoading(false);
      }
    };
  });
}
```

20. Custom react hook

- Custom Hook 으로 http function 이용해보기.
> /src/hooks/useFetch.js (2 / 2)

```
    fetchCall();  
  }, [fnFetch]);  
  
  return { fetchedData, setFetchedData, isLoading, error };  
}
```

20. Custom react hook

- useFetch 이용해보기

> /src/App.js (1 / 2)

-- 생략 --

```
import { useFetch } from "../hooks/useFetch";
```

```
function App() {
```

-- 생략 --

```
const [todoLists, todoDispatcher] = useReducer(taskReducers, []);
```

```
const { fetchedData, isLoading } = useFetch(todoLists, loadTasks);
```

```
useEffect(() => {
```

```
  if (!isLoading) {
```

```
    todoDispatcher({ type: actionTypes.init, payload: fetchedData.body });
```

```
  }
```

```
}, [fetchedData, isLoading]);
```

```
// const [isLoading, setIsLoading] = useState(true);
```

```
// const taskFetch = async () => {
```

```
//   setIsLoading(true);
```

```
//   try {
```

20. Custom react hook

- useFetch 이용해보기

> /src/App.js (2 / 2)

```
// const json = await loadTasks();  
// todoDispatcher({ type: actionType.init, payload: json.body });  
// } catch (e) {  
// alert(e.message || "데이터 조회 실패");  
// } finally {  
// setIsLoading(false);  
// }  
// };  
// useEffect(() => {  
// taskFetch();  
// }, []);
```

```
const taskCount = useMemo(() => {  
  -- 생략 --  
}, [todoLists]);  
  
-- 생략 --  
}
```

React Router

- 21. React Router
 - 22. Route
 - 23. Link
 - 24. Nested Route
- 25. Route error handling
 - 26. Dynamic Route
 - 27. Index Route

21. React Router

21. React Router

- React 는 SPA(Single Page Application) Framework
 - 한 페이지에서 여러 개의 컴포넌트를 제어해, 여러 화면을 보는 것처럼 구성할 수 있다.
 - 단 웹 브라우저의 뒤로 가기, 앞으로 가기 등의 History를 관리할 수 없는 환경.
- React Component의 History를 관리하기 위해 React-Router를 사용.
 - 하나의 index.html 페이지에서 여러 컴포넌트 화면을 자연스럽게 이동할 수 있다.
 - 또한, URL 을 기반으로 컴포넌트를 변경하기 때문에 History관리도 가능하다.

21. React Router

- React Router 설치
 - `npm install react-router-dom`

22. Route

22. Route

- React Router를 이용하기 위해 Route를 먼저 정의해야 한다.

```
const routers = createBrowserRouter([
  {
    path: "URL",
    element: <노출할 컴포넌트 />,
  }, {
    path: "URL",
    element: <노출할 컴포넌트 />,
  },
]);
```

22. Route

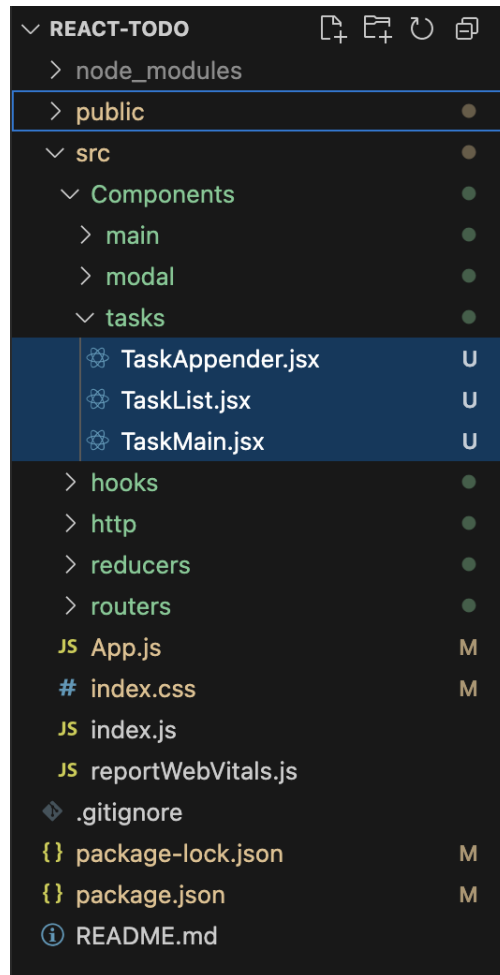
- 메인 페이지 생성하기

> /src/Components/main/Main.jsx

```
export default function Main() {  
  return <section className="wrapper">Welcome to React Todo!</section>;  
}
```

22. Route

- App.js 내용을 /Components/tasks/TaskMain.jsx 만들어 이동시키기
- TaskList.jsx, TaskAppender.jsx 모두 /Components/tasks 로 이동시키기



22. Route

- Router 만들기

> /src/routers/RouterAppProvider.js

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import Main from "../Components/main/Main";
import TaskMain from "../Components/tasks/TaskMain";

export default function RouterAppProvider() {
  const routers = createBrowserRouter([
    { path: "/", element: <Main /> },
    { path: "/todo", element: <TaskMain /> },
  ]);

  return <RouterProvider router={routers} />;
}
```

22. Route

- Router 적용하기

> /src/App.js

```
import RouterAppProvider from "../routers/router";

function App() {
  return <RouterAppProvider />;
}

export default App;
```

23. Link

23. Link

- 브라우저에서 확인해보기
 - <http://localhost:3000/>
 - <http://localhost:3000/todo>
- URL이 변경될 때 마다 Router에 의해 알맞은 컴포넌트가 출력된다.
- React는 페이지가 새롭게 열릴 때마다 state가 초기화 되기 때문에
- URL로 분리해서 접속하면 안됨.
- React 전용 Anchor(<a>)인 Link 컴포넌트를 이용해야 한다.
 - Link 컴포넌트는 RouterProvider 내부에서만 동작이 가능하다.

23. Link

- Link를 제공할 HeaderNav 컴포넌트 만들기

> /src/Components/layout/HeaderNav.js

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

export default function HeaderNav() {
  return (
    <header>
      <nav className="menu-navigation">
        <ul>
          <li>
            <Link to="/">Main</Link>
          </li>
          <li>
            <Link to="/todo">Task</Link>
          </li>
        </ul>
      </nav>
    </header>
  );
}
```

24. Nested Route

24. Nested Route

- 라우터 Path별로 HeaderNav를 제공하기 위해 중첩 라우터를 이용
- 먼저 HeaderNav를 표현하기 위한 MainLayout을 생성

> /src/Components/layout/MainLayout.jsx

```
import { Outlet } from "react-router-dom";
import HeaderNav from "../HeaderNav";

export default function MainLayout() {
  return (
    <div className="main-container">
      <HeaderNav />
      <Outlet />
    </div>
  );
}
```

- <Outlet> 컴포넌트는 중첩 라우터를 노출시킨다.
 - URL에 따라 <Main />, <TaskMain /> 컴포넌트가 노출된다.

24. Nested Route

- Router를 중첩라우터로 변경한다.
> /src/routers/RouterAppProvider.js

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import Main from "../Components/main/Main";
import TaskMain from "../Components/tasks/TaskMain";
import MainLayout from "../Components/layout/MainLayout";

export default function RouterAppProvider() {
  const routers = createBrowserRouter([
    {
      path: "/",
      element: <MainLayout />,
      children: [
        { path: "", element: <Main /> },
        { path: "todo", element: <TaskMain /> },
      ],
    },
  ]);

  return <RouterProvider router={routers} />;
}
```

24. Nested Route

- 현재 URL에 따라 메뉴를 강조 시키는 NavLink로 변경

> /src/Components/layout/HeaderNav.jsx

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

export default function HeaderNav() {
  return (
    <header>
      <nav className="menu-navigation">
        <ul>
          <li>
            <NavLink to="/">Main</NavLink>
          </li>
          <li>
            <NavLink to="/todo">Task</NavLink>
          </li>
        </ul>
      </nav>
    </header>
  );
}
```

25. Route error handling

25. Route error handling

- Router에서 제공하지 않는 URL로 직접 접근할 경우
- (<http://localhost:3000/articles>) Router Error 페이지가 노출된다.

Unexpected Application Error!

404 Not Found

 Hey developer 🙌

You can provide a way better UX than this when your app throws errors by providing your own `ErrorBoundary` or `errorElement` prop on your route.

- Router Error 페이지가 아닌 Custom Error 페이지로 변경할 수 있다.

25. Route error handling

- Custom Error 컴포넌트 만들기

> /src/Components/layout/NotFound.jsx

```
import HeaderNav from "./HeaderNav";

export default function NotFound() {
  return (
    <div className="main-container">
      <HeaderNav />
      <div className="wrapper">페이지를 찾을 수 없습니다.</div>
    </div>
  );
}
```

25. Route error handling

- Error Router 연결하기

> /src/routers/RouterAppProvider.jsx

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import Main from "../Components/main/Main";
import TaskMain from "../Components/tasks/TaskMain";
import MainLayout from "../Components/layout/MainLayout";
import NotFound from "../Components/layout/NotFound";

export default function RouterAppProvider() {
  const routers = createBrowserRouter([
    {
      path: "/",
      element: <MainLayout />,
      errorElement: <NotFound />,
      children: [
        { path: "", element: <Main /> },
        { path: "todo", element: <TaskMain /> },
      ],
    },
  ]);

  return <RouterProvider router={routers} />;
}
```

25. Route error handling

- 존재하지 않는 Path로 접근하면 NotFound 컴포넌트가 노출된다.

Main Task

페이지를 찾을 수 없습니다.

26. Dynamic Route

26. Dynamic Route

- TaskItem을 클릭하면, 상세 Task 정보가 나오도록 개선해보기

> /src/Components/tasks/TaskList.jsx > TaskItem

```
import { createContext, memo, useContext } from "react";
import { Link } from "react-router-dom";

-- 생략 --

function TaskItem({ done, id, task, dueDate, priority, onCheckboxClick }) {
  -- 생략 --
  return (
    <li className="task-item">
      <input
        -- 생략 --
      />
      <label htmlFor={id} className={done ? "done-todo" : undefined}>
        <Link to={`/todo/${id}`}>{task}</Link>
      </label>
      <span className={`due-date ${done ? "done-todo" : undefined}`}>
        {dueDate}
      </span>
      <span className={`priority ${done ? "done-todo" : undefined}`}>
        {priority}
      </span>
    </li>
  );
}
```

26. Dynamic Route

- Task 정보 받아오는 http 함수 생성

> /src/http/taskHttp.js (1 / 2)

-- 생략 --

```
export async function getTask(taskId) {  
  const response = await fetch(`http://localhost:8888/api/v1/task/${taskId}`);  
  
  const json = await response.json();  
  if (json.status === 200) {  
    return json;  
  }  
  
  throw new Error(response.statusMessage);  
}
```

-- 생략 --

26. Dynamic Route

- TaskItem 컴포넌트 생성

> /src/Components/tasks/TaskDetail.jsx (1 / 3)

```
import { useParams } from "react-router-dom";
import { useFetch } from "../../hooks/useFetch";
import { doneTask, getTask } from "../../http/taskHttp";
import { useCallback, useState } from "react";

export default function TaskItem() {
  const param = useParams();
  const id = param.id;

  // 컴포넌트 새로고침용.
  const [rnd, setRnd] = useState();

  const taskFetch = useCallback(() => {
    return getTask(id);
  }, [id, rnd]);

  const { fetchedData, isLoading } = useFetch({}, taskFetch);

  const doneTodoItemHandler = () => {
```

26. Dynamic Route

- TaskItem 컴포넌트 생성

> /src/Components/tasks/TaskDetail.jsx (2 / 3)

```
const doneFetch = async (fnCallback) => {  
  try {  
    const json = await doneTask(id);  
    fnCallback(json.body.taskId);  
  } catch (e) {  
    alert(e.message || "task 완료 실패");  
  }  
};
```

```
doneFetch(() => {  
  setRnd(Math.random());  
});  
};
```

```
return (  
  <div className="wrapper">  
    {isLoading && <div>Loading...</div>}  
    {!isLoading && (  
      <div>
```


26. Dynamic Route

- TaskItem 컴포넌트 생성

> /src/Components/tasks/TaskDetail.jsx (3 / 3)

```
<h1>{fetchData.body.task}</h1>
<h3>완료 예정일자: {fetchData.body.dueDate}</h3>
<h3>우선순위: {fetchData.body.priority}</h3>
<h3>등록일자: {fetchData.body.createAt}</h3>
{!fetchData.body.done && (
  <button
    type="button"
    className="confirm-ok"
    onClick={doneTodoItemHandler}
  >
    완료
  </button>
)}
{fetchData.body.done && (
  <h3>완료일자: {fetchData.body.doneAt}</h3>
)}
</div>
)}
</div>
);
}
```

26. Dynamic Route

- Router에 TaskItem 등록

> /src/routers/RouterAppProvider.jsx (1 / 2)

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import Main from "../Components/main/Main";
import TaskMain from "../Components/tasks/TaskMain";
import TaskItem from "../Components/tasks/TaskDetail";
import MainLayout from "../Components/layout/MainLayout";
import NotFound from "../Components/layout/NotFound";

export default function RouterAppProvider() {
  const routers = createBrowserRouter([
    {
      path: "/",
      element: <MainLayout />,
      errorElement: <NotFound />,
      children: [
        { path: "", element: <Main /> },
        { path: "todo", element: <TaskMain /> },
        { path: "todo/:id", element: <TaskItem /> },
      ],
    },
  ]),
}
```

26. Dynamic Route

- Router에 TaskItem 등록

> /src/routers/RouterAppProvider.jsx (2 / 2)

```
]);  
  
return <RouterProvider router={routers} />;  
}
```

27. Index Route

27. Index Route

- Router 중 Path가 공백으로 된 Route는 Index Route로 사용가능.

> /src/routers/RouterAppProvider.jsx

-- 생략 --

```
export default function RouterAppProvider() {
  const routers = createBrowserRouter([
    {
      path: "/",
      element: <MainLayout />,
      errorElement: <NotFound />,
      children: [
        { path: "", element: <Main /> },
        { path: "todo", element: <TaskMain /> },
        { path: "todo/:id", element: <TaskItem /> },
      ],
    },
  ]);

  return <RouterProvider router={routers} />;
}
```

← Path가 없는 Router들은 대표 Router가 될 수 있다.

27. Index Route

- Path가 "" 인 Route는 모두 index Route로 변경

> /src/routers/router.js

-- 생략 --

```
export default function RouterAppProvider() {
  const routers = createBrowserRouter([
    {
      path: "/",
      element: <MainLayout />,
      errorElement: <NotFound />,
      children: [
        { index: true, element: <Main /> },
        { path: "todo", element: <TaskMain /> },
        { path: "todo/:id", element: <TaskItem /> },
      ],
    },
  ]);

  return <RouterProvider router={routers} />;
}
```

감사합니다.

최신 React.js로 구현하는
프론트엔드 애플리케이션 개발

장민창

mcjang1116@gmail.com