

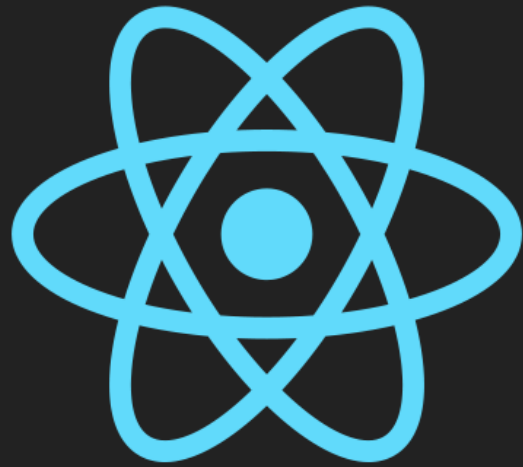
Hello,

KDT 웹 개발자 양성 프로젝트

1기! 40th

with





HOOKS

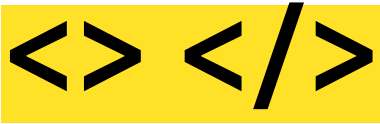


그래서 탄생, React HOOKS

- 다만 기존 클래스형 컴포넌트에서 사용하던 편리한 기능(리액트의 핵심)을 함수형에 적용하려니 기존 것을 그대로 사용할 수는 없었고 새로운 것이 필요 했습니다
- 그래서 탄생한 것이 React HOOKS 입니다
- 앞에 use 가 붙은 애들이 HOOKS 들이죠
 - useState / useRef / useEffect / useContext / useMemo / useCallback / useReducer



React.Fragment



- 개발자들은 축약의 만족이기 때문에 이렇게 긴 코드를 용납 못합니다!
- `<React.Fragment>` 는 `<>` 로 대체가 가능합니다! :)

```
import React from "react";

export default function ReactFragment() {
  return (
    <>
      <h1>안녕하세요!</h1>
      <span>반갑습니다!</span>
    </>
  );
}
```

src/components/ReactFragment.js



React.Fragment

가 필요할 때!



useState

useRef

Variable

State

변경



Ref

변경



Variable

변경

?





Life Cycle

componentDidMount

componentDidUpdate

componentWillUnmount



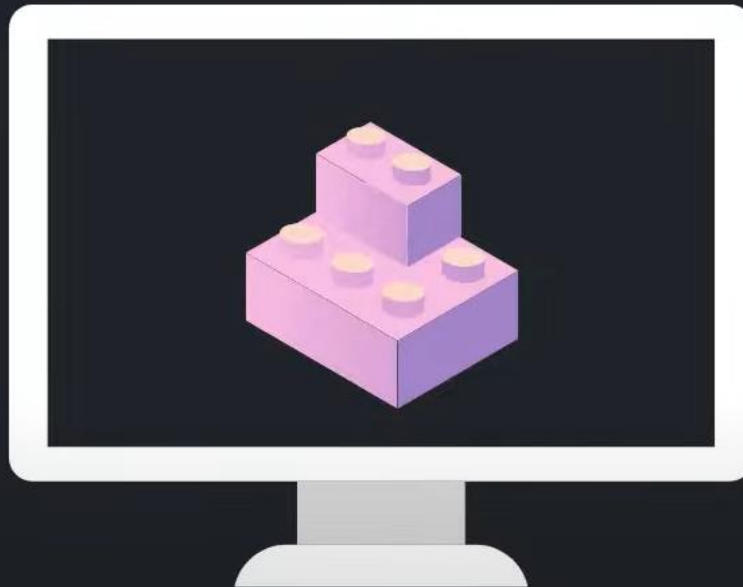
Mount

화면에 첫 렌더링



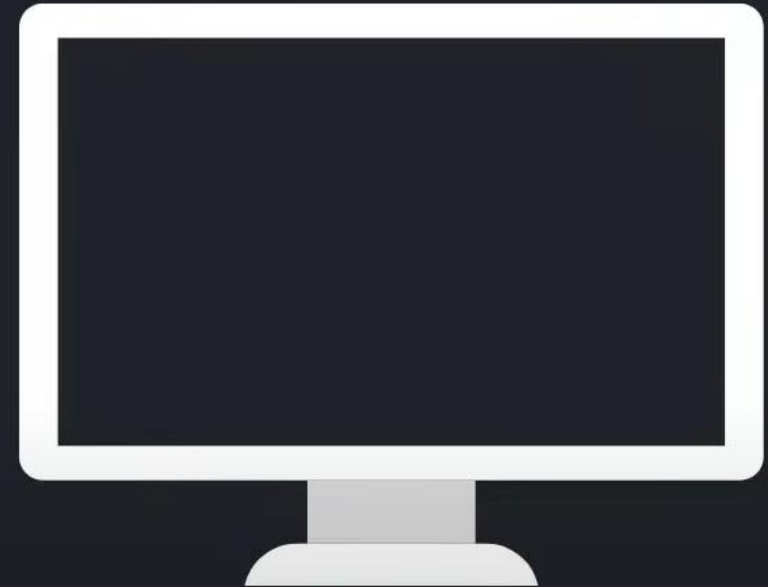
Update

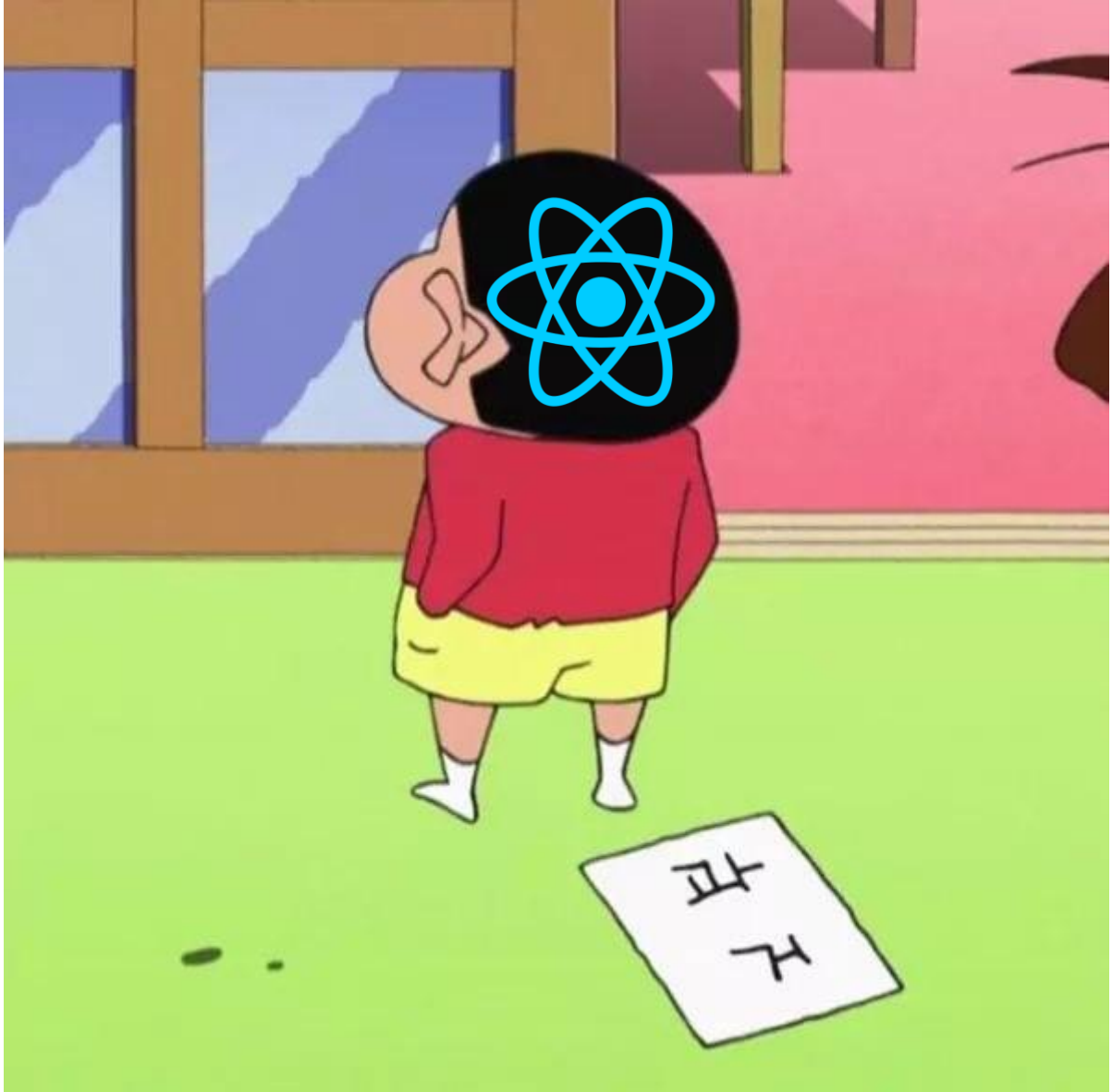
다시 렌더링



Unmount

화면에서 사라질때





Mount

화면에 첫 렌더링



Update

다시 렌더링



Unmount

화면에서 사라질때



useEffect



useEffect



useEffect



useEffect

별코딩★

<https://www.youtube.com/watch?v=kyodvzc5GHU>

1

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

```
  // 작업...
```

```
});
```

렌더링 될때 마다 실행

2

```
useEffect( ( ) => {  
    // 작업...  
}, [ value ] );
```

화면에 첫 렌더링 될때 실행

value 값이 바뀔때 실행

Clean Up - 정리



```
useEffect( ( ) => {  
    // 구독 ...  
  
    return ( ) => {  
        // 구독 해지 ...  
    }  
}, [ ] );
```




useState
useRef
useEffect

세상에서 제일 중요한 거거든

useMemo



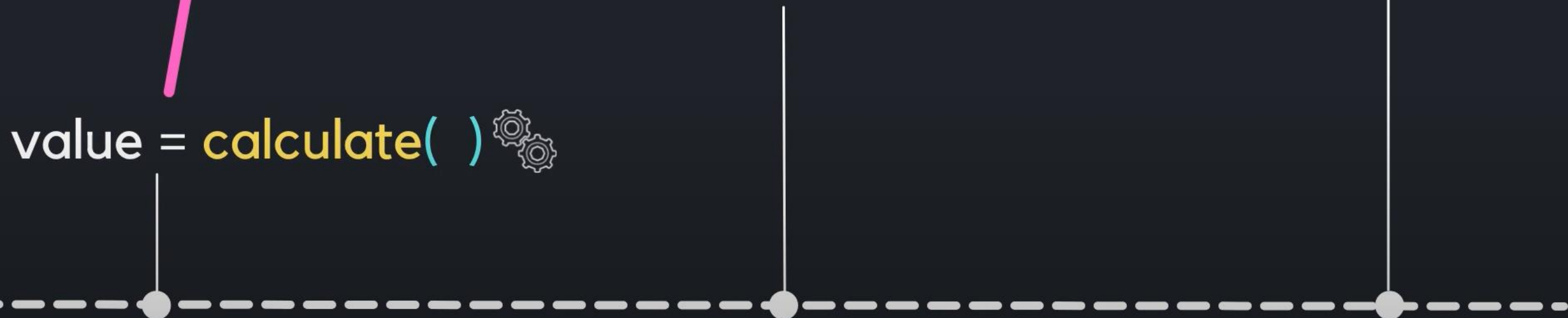


value = calculate() 



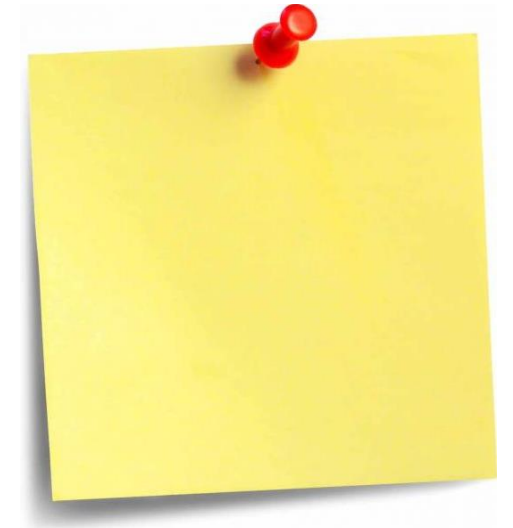
10
value

10
value



useMemo

실전 활용!





useMemo 실전 활용

- 이 상태에서는 이전과는 달리 숫자를 변경해도 useEffect 가 호출이 됩니다!
- 그리고 useEffect 에 노란색 줄이 가 있네요?

```
(property) where: string
```

The 'location' object makes the dependencies of useEffect Hook (at line 13) change on every render. To fix this, wrap the initialization of 'location' in its own useMemo()

Hook. eslint([react-hooks/exhaustive-deps](#))

문제 보기 빠른 수정... (Ctrl+.)

```
... where: isKorea ? "한국" : "외국",  
... }
```

원시 (Primitive) 타입

String
Number
Boolean
Null
Undefined
BigInt
Symbol

객체 (Object) 타입

원시 타입을 제외한 모든 것

Object
Array
...

원시 (Primitive) 타입

```
const locationOne = "korea"
```

```
const locationTwo = "korea"
```

```
locationOne === locationTwo
```

```
> true
```

객체 (Object) 타입

```
const locationOne = {  
  country: "korea"  
}
```

```
const locationTwo = {  
  country: "korea"  
}
```

```
locationOne === locationTwo
```

```
> false
```



```
export default function UsingUseMemo() {  
  const [number, setNumber] = useState(0);  
  const [isKorea, setIsKorea] = useState(true);  
  
  const location = useMemo(() => {  
    return {  
      where: isKorea ? "한국" : "외국",  
    }  
  }, [isKorea]);  
  
  useEffect(() => {  
    console.log("✨ useEffect 호출!");  
  }, [location])  
}
```

src/components/UsingUseMemo.js



**편한게
짱이야!**



편리미엄

소비자들이 가격이나 품질 등 가성비를 넘어
시간과 노력을 아낄 수 있는
편리한 상품이나 서비스를 선호하는 현상



위 이미지는 시선뉴스에서 자체 제작한 콘텐츠로 타 매체에서의 무단 전재·복사·배포 등을 금지합니다. (www.sisunnews.co.kr 시선뉴스)





ES7+ React/Redux/React-Native snippets

dsznajder | 6,276,404 | ★★★★★ (62)

Extensions for React, React-Native and Redux in JS/TS with ES7+ syntax. Cu...

사용 안 함

제거



이 확장은 전역적으로 사용하도록 설정되었습니다.

세부 정보

기능 기여도

변경 로그

런타임 상태



rfc

<input type="checkbox"/>	rfc	reactFunctionalComponent
<input type="checkbox"/>	rfce	reactFunctionalExportComponent
<input type="checkbox"/>	rfcp	reactFunctionalComponentWithPropTypes
<input type="checkbox"/>	rfcredux	reactFunctionalComponentRedux
<input type="checkbox"/>	rfcreduxp	reactFunctionalComponentReduxPropTypes
<input type="checkbox"/>	raf	reactArrowFunctionComponent
<input type="checkbox"/>	rafce	reactArrowFunctionExportComponent
<input type="checkbox"/>	rafcp	reactArrowFunctionComponentWithPropTypes
	ReactFragment	./ReactFragment
	RTCDTMFToneChangeEvent	
	Reflect	
	RTCDTMFSender	



src > components > JS FancyBorder.js > ...

```
1 import React from 'react'
2
3 export default function FancyBorder() {
4   return (
5     <div>FancyBorder</div>
6   )
7 }
8
```





JSX 와

IF문!



JSX 와 IF문을 조합해 볼까요?

- 조건부 렌더링의 경우 상황에 따라서 해당 컴포넌트를 보여 줄지 말지 여부를 결정합니다!
- 삼항 연산자를 if 문으로 대체해도 문제가 없지만 아무래도 코드가 더 짧고, 직관적(?)인 3항 연산자 또는 && 연산자가 더 많이 쓰입니다!
- 그럼 IF문은 쓸 일이 없을까요?



```
export default function Where() {  
  const where = prompt("어디로 갈까요? left / right")  
  
  if (where === "left") {  
    return (  
      <>  
        <h1>여기는 왼쪽입니다!</h1>  
      </>  
    )  
  } else {  
    return (  
      <>  
        <h1>여기는 오른쪽 입니다!</h1>  
      </>  
    )  
  }  
}
```

src/components/Where.js



이정도면?

- 저 정도 코드면 3행 연산자가 더 편하지 않을까요?

```
export default function Where() {  
  const where = prompt("어디로 갈까요? left / right")  
  
  return (  
    <>  
    {where === "left" ? <h1>이쪽은 왼쪽입니다</h1> : <h1>이쪽은 오른쪽 입니다</h1>}  
    </>  
  )  
}
```

src/components/Where.js

- 그럼 아래의 코드를 보시죠!



```
export default function BoardDetail() {
  const { boardID } = useParams();
  const location = useLocation();
  const articleList = [1, 2];

  if (articleList.find((el) => el === parseInt(boardID))) {
    return (
      <>
        <Header />
        <h2>{boardID} 번 게시물 내용입니다!</h2>
        <p>쿼리: {location.search}</p>
        <p>주소: {location.pathname}</p>
        <p>해쉬: {location.hash}</p>
      </>
    );
  } else {
    return (
      <>
        <NotFound />
      </>
    )
  }
}
```



IF문의 사용

- 위와 같이 조건식 자체가 복잡하거나, 조건에 따른 결과 리턴이 완전히 달라지는 경우에는 IF문을 사용하는 편이 더 보기에 편하고 처리 역시도 편합니다!



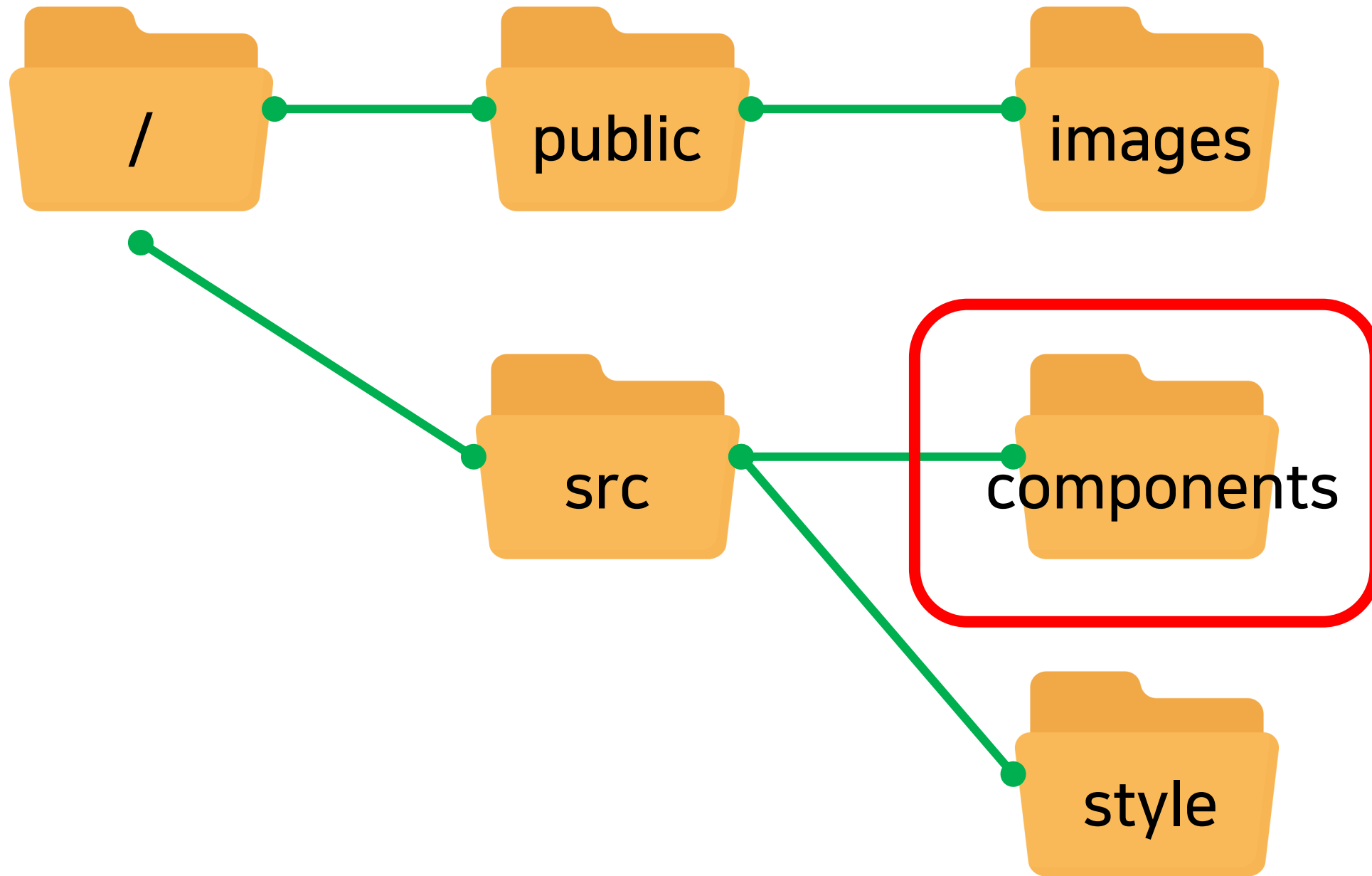
Public

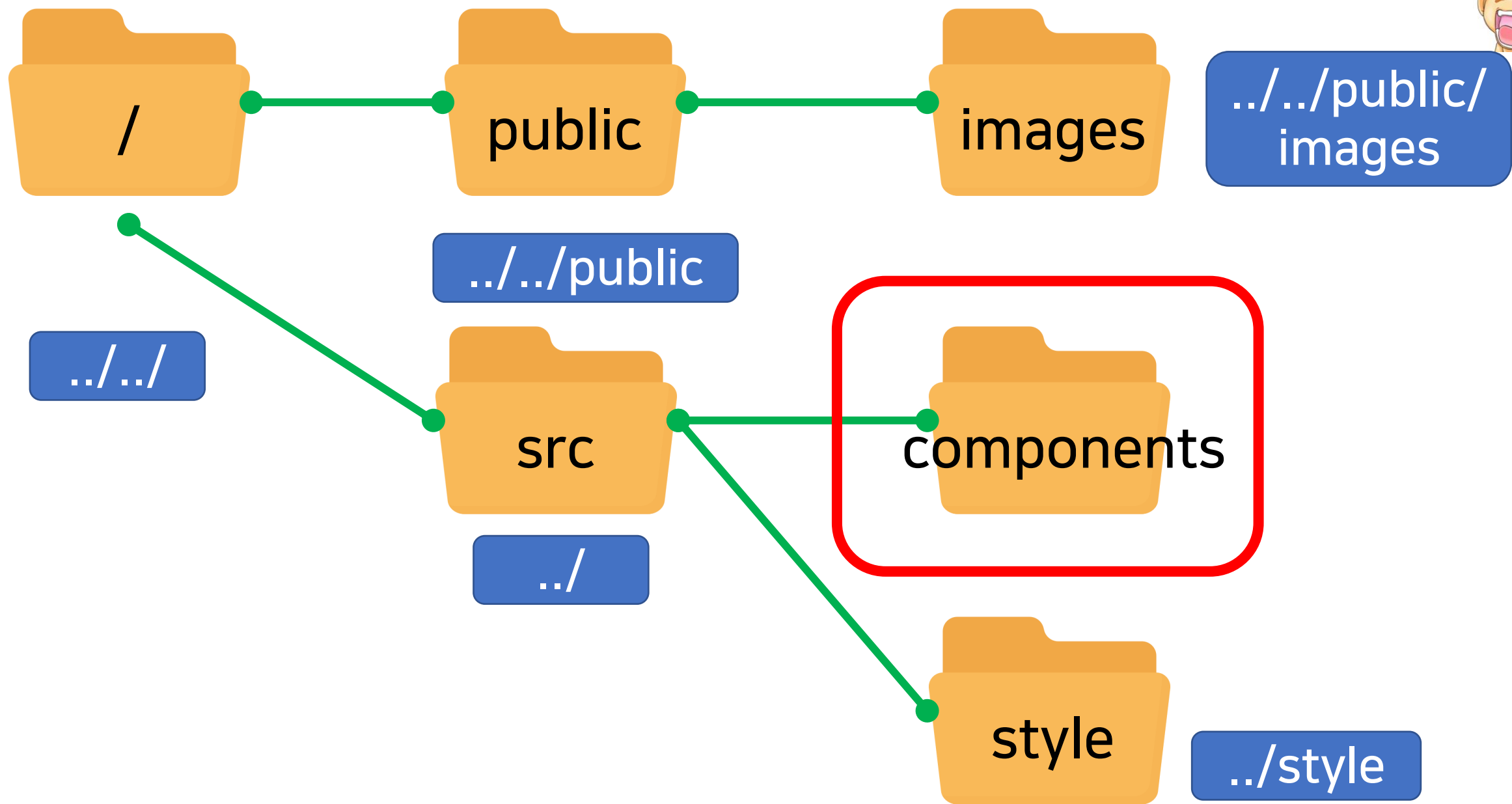
폴더 사용



퍼블릭 폴더로 접근하기!

- 기존 Backend 에서는 public 폴더를 static 이라는 Express 메소드를 사용해서 특정 주소 값을 요청하면 바로 public 폴더로 연결 해줬습니다!
- Public 폴더에 **/images** 라는 폴더를 만들고 원하는 사진 한 장을 넣기!
- 간단하게 **<Image>** 라는 컴포넌트를 만들고 public 폴더에 있는 이미지 파일에 접근해 봅시다!







상대 경로로 과연 가질까요!?

- 위에서 표현한 접근 방법으로 표시해서 접근해 봅시다!

```
import dogImg from "../../public/images/dog.jpg"

export default function Image() {
  return (
    <>
      <img src={dogImg} alt="강아지" />
    </>
  )
}
```

src/components/Image.js



Compiled with problems:

ERROR in ./src/components/test/Image.js 4:0-52

Module not found: Error: You attempted to import ../../../../public/images/dog.jpg which falls outside of the project src/ directory. Relative imports outside of src/ are not supported.
You can either move it inside src/, or add a symlink to it from project's node_modules/.

- Error 를 읽어보니 컴포넌트에서 다른 폴더에 접근을 하려고 해도 src 폴더 밖으로 나가는 것은 지원하지 않는다고 하네요
- 그럼, 백엔드 처럼 static 설정하면 되게겠죠?



잠깐! 그럼 절대 경로는!?

```
import dogImg from "/"

export default function() {
  return (
    <img src=
  )
}
```

- \$Recycle.Bin
- \$WINDOWS.~BT
- \$WinREAgent
- \$Windows.~WS
- Documents and Settings
- ESD
- Microsoft VS Code
- PerfLogs
- Program Files
- Program Files (x86)
- ProgramData
- Recovery

- 진짜 컴퓨터의 Root 경로가 뜨네요
- 이건 아무리 봐도 답이 없겠죠?
- 그럼 Static 설정을 해봅시다!





폐북의 가호

- `Npx create-react-app` 을 통해 만들어진 리액트 앱의 경우는 public 폴더가 자동으로 static 처리가 됩니다!
- 따라서, 어느 위치에서건 `/` 를 써서 접근하면 public 폴더가 호출 됩니다!
- 그럼 강아지 사진의 주소 값을 `/images/dog.jpg` 로 변경해 봅시다!



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

src/components/Image.js





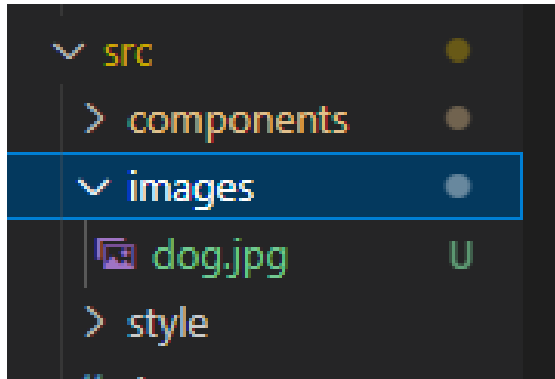
예전에 말씀드린 보안적 측면!

- 주소를 통해 폴더 구조가 드러나는 경우를 막기 위해 static 처리를 하고 있습니다!
- 상대 경로를 이용해서 src 폴더 이상으로 가서 다른 폴더에 접근해서 다시 내려가는 방식은 막힙니다!
- 또는, 절대 경로를 이용하는 방법 역시도 막힙니다!
- 단, 한단계 상위 폴더로 접근하는 상대 경로는 먹힙니다!(아마도 편의를 위해서 그런 것 같네요)



하지만 src 폴더 이내라면!?

- 아무래도 src 폴더 내부에서는 서로가 서로를 참조하는 일이 비일비재 할 것 이다보니 아까 경고문에서도 src 외부로 나가는 것을 지원하지 않는다고 한 것 같네요!
- 그럼, 강아지 사진을 `src/images` 폴더로 옮기고 테스트 해봅시다!



```
import dogImg from "../images/dog.jpg"

export default function Image() {
  return (
    <>
    <img src={dogImg} alt="강아지" />
    </>
  )
}
```

src/components/Image.js

src 폴더 내부 참조는 문제 없습니다!





Component Composition



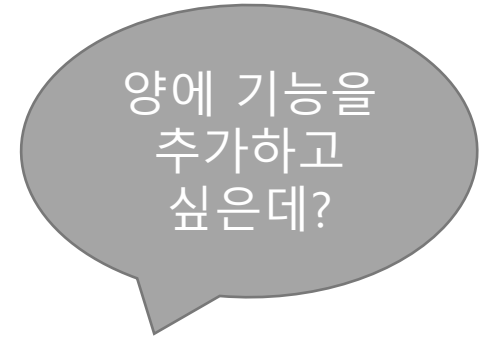
Specialization

Class 의 상속 확장



양
CLASS

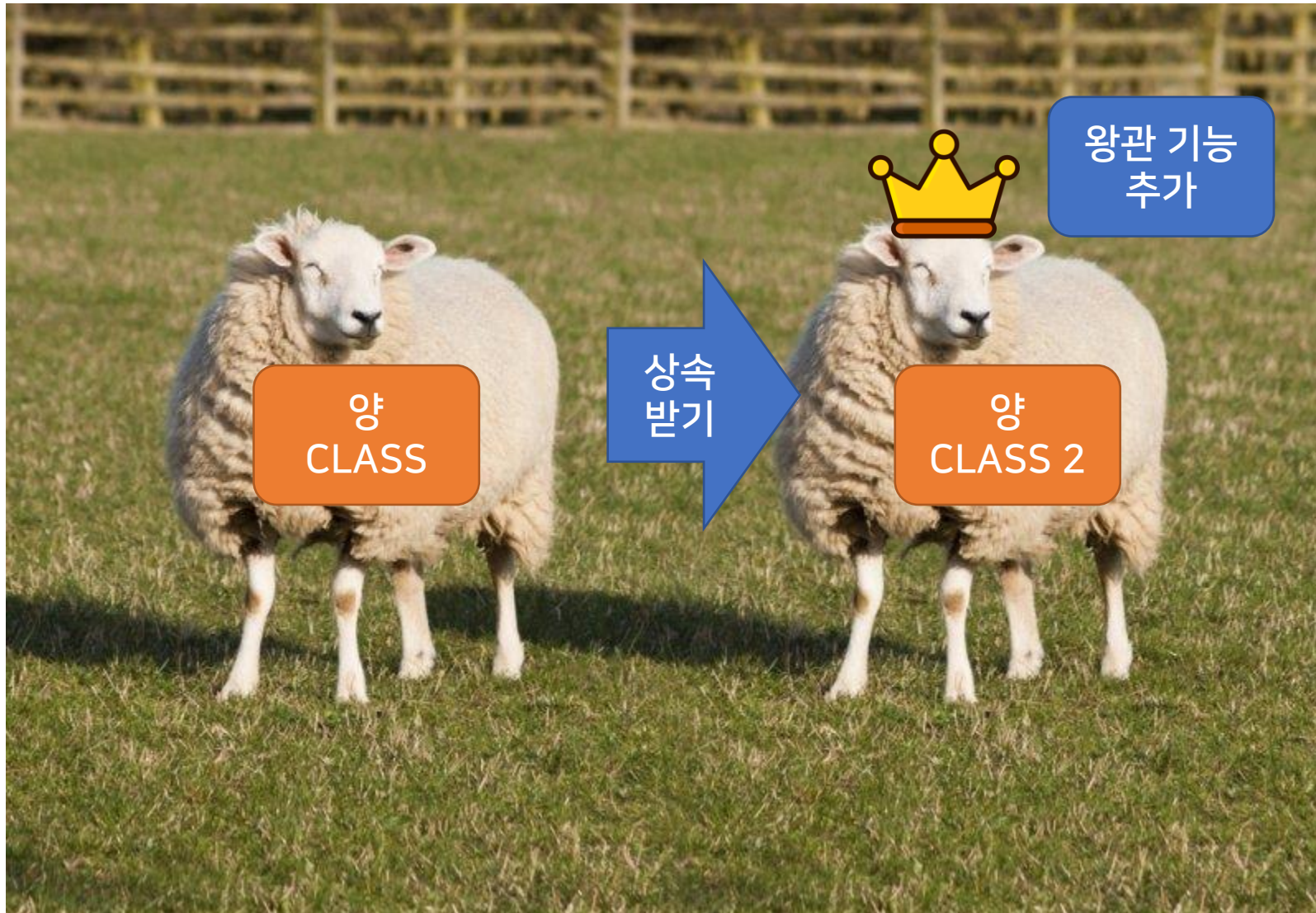
Class 의 상속 확장



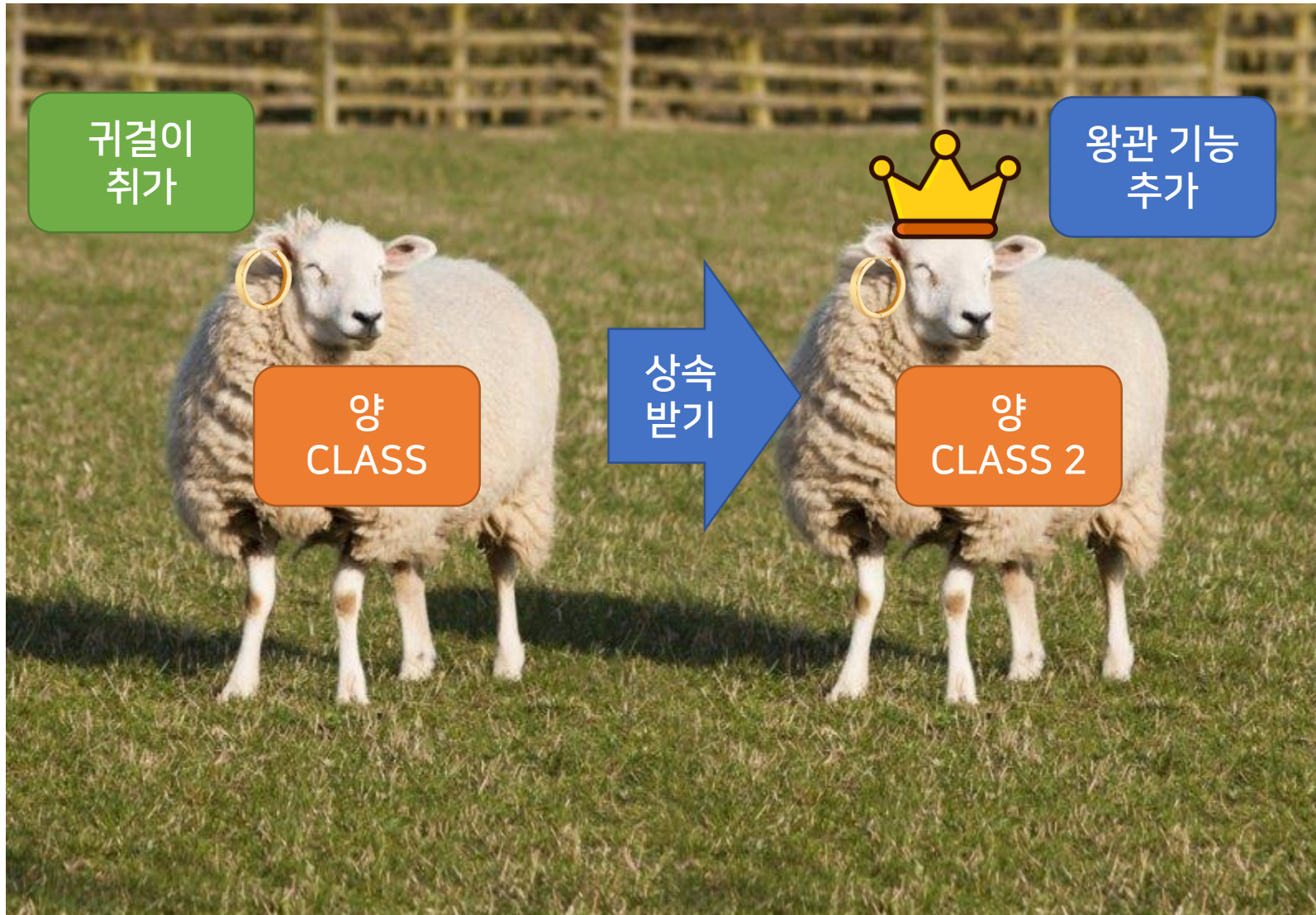
Class 의 상속 확장



Class 의 상속 확장



Class 의 상속 확장





Composition(합성)

- React 에서는 합성(Composition)이라는 방법을 제공합니다!
- 기존 JS 에서는 특정 기능 또는 역할을 담당하는 것을 확장하여 사용하기 위해서는 Class 를 사용해야만 하며, 해당 Class 를 상속(extends) 받아서 확장 해야만 기능의 추가 및 활용이 가능하였습니다!



Composition(합성)

- 리액트에서는 컴포넌트 단위로만 구성을 하면 되기 때문에 Class 의 확장과 같은 복잡한 개념(생성자, super 등등) 보다는 블록 조립에 가까운 합성의 개념을 도입 했습니다
- 기존의 것이 필요하면 import 로 불러와서 바로 사용하고, 필요한 기능을 추가한 뒤 새로이 명명하는 방법으로 확장과 비슷한 개념을 더 단순화 하여 사용하고 있습니다
- 그리고 각각의 커스터마이징은 Props 를 통해 간단히 구현 가능합니다!



React 의 합성(Composition)~!



- 웹 페이지에 간단한 문구를 띄워주는 역할



React 의 합성(Composition)~!



- 웹 페이지에 간단한 문구를 띄워주는 역할
- 사용자에게 Welcome 메시지를 띄워 주는 역할 추가
- 그리고 사용자가 원하는 요소를 추가하여 새로운 형태의 컴포넌트 만들기



React 의 합성(Composition)~!



- 웹 페이지에 간단한 문구를 띄워주는 역할
- 사용자에게 Welcome 메시지를 띄워 주는 역할 추가
- 그리고 사용자가 원하는 요소를 추가하여 새로운 형태의 컴포넌트 만들기

→ Props 를 통해 원하는 것 전달

→ WelcomeDialog 라는 컴포넌트로 명명





컴포넌트 만들기

- 간단한 형태의 `<Dialog>` 라는 컴포넌트를 만들어 봅시다!
- 해당 컴포넌트는 props 로 title, message, color 를 받아서 화면에 출력해 주는 컴포넌트 입니다!



```
export default function Dialog(props) {  
  return (  
    <div style={{ backgroundColor: props.color }}>  
      <h1>{props.title}</h1>  
      <p>{props.message}</p>  
    </div>  
  );  
}
```

src/components/Dialog.js

```
function App() {  
  return (  
    <div className="App">  
      <Dialog  
        color="orange"  
        title="Welcome to summoner's lift"  
        message="소환사의 협곡에 오신 걸 환영합니다."  
      />  
    </div>  
  );  
}
```

src/App.js



Welcome to summoner's lift

소환사의 협곡에 오신 걸 환영합니다.



Specialization



컴포넌트 Specialization

- <Dialog> 의 배경 색상은 orange 이고 h1 태그에는 Welcome to summoner's lift, p 태그에는 소환사의 협곡에 오신 것을 환영합니다 라는 메시지를 전달하는 dialog 를 자주 사용하게 되었다고 가정해 봅시다!



컴포넌트 Specialization

- 그럴때 마다 <Dialog> 컴포넌트에 props 로 color, title, message 를 전달하는 방법이 좋을까요?
- 아니면 원하는 구조의 컴포넌트를 만들어 놓고 재 사용하는 방법이 좋을까요?
- 그런데 이미 우리는 <Dialog> 라는 컴포넌트가 있는데요? 이를 재활용 해봅시다!



컴포넌트 Specialization

- `<Dialog>` 를 불러와서 사용하는 `<WelcomeDialog>` 컴포넌트를 구성!
- 다시 처음부터 `<Dialog>` 를 그리는 것이 아니라 `<WelcomeDialog>` 에서 `<Dialog>` 를 불러오고 원하는 props 만 전달 하는 것(App.js 에서 하던 것)을 하나의 컴포넌트로 만들고 이를 편하게 불러서 사용합시다!



```
import Dialog from "../Dialog";

export default function WelcomeDialog() {
  return (
    <Dialog
      color="orange"
      title="Welcome to sommoner's lift"
      message="소환사의 협곡에 오신 걸 환영합니다."
    />
  );
}
```

src/components/WelcomeDialog.js

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

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

src/App.js



Welcome to summoner's lift

소환사의 협곡에 오신 걸 환영합니다.



- 매번 사용되는 컴포넌트와 특정 props 값이 있다면 아예 그것을 선언함 컴포넌트를 만들어 재사용!



실습, WelcomeDialog 특수화!

- <WelcomeDialog> 컴포넌트에 button 요소를 추가해 주세요
- 버튼은 클릭하면 alert 창을 띄웁니다
- 버튼 요소의 content 와 alert 창 의 메시지는 각각 props.content 와 props.alertMessage 로 전달을 받습니다!
- Props.content 로 항상 “클릭해 주세요!” 와 props.alertMessage 로 “클릭해 주셨군요!” 를 전달하는 특수화 된 <WelcomeDialogBtn> 컴포넌트를 만들어 주세요!



Props.children



Props.children

- Props 로 값을 보내는 것은 이미 배우셨습니다!
- 다만, 이미 정해진 props에 전달하는 값이 아닌 상황에 따라 html 요소 또는 컴포넌트 자체를 보내고 싶을 땐 어떻게 하면 될까요?
- 이럴 때에는 컴포넌트의 자식 요소를 한꺼번에 전달해 주는 `props.children` 을 사용할 수 있습니다.



```
React.createElement(  
  type,  
  [props],  
  [...children]  
);
```



```
1 function WelcomeDialog(props) {  
2   return (  
3     <FancyBorder color="blue">  
4       <h1 className="Dialog-title">  
5         어서오세요  
6       </h1>  
7       <p className="Dialog-message">  
8         우리 사이트에 방문하신 것을 환영합니다!  
9       </p>  
10    </FancyBorder>  
11  );  
12 }
```

Props

Props.children



코드로 확인해 봅시다!

- 전달 받은 요소를 전달 받은 color 의 border 로 감싸는 `<FancyBorder>` 라는 컴포넌트를 작성하여 봅시다!
- 그리고 App.js 에서 해당 컴포넌트가 감쌀 요소를 `props.children` 으로 한 꺼번에 전달해 봅시다!



```
export default function FancyBorder(props) {  
  return (  
    <div style={{ border: `3px solid ${props.color}` }}>  
      {props.children}  
    </div>  
  );  
}
```

src/components/FancyBorder.js

```
import FancyBorder from "../components/FancyBorder";  
  
function App() {  
  return (  
    <div className="App">  
      <FancyBorder color="blue">  
        <h1>Hello, props.children</h1>  
        <p>이건 매우 유용한 기술입니다요!</p>  
      </FancyBorder>  
    </div>  
  );  
}
```

src/App.js



Hello, props.children

이건 매우 유용한 기술입니다요!



```
export default function FancyBorder(props) {  
  return (  
    <div style={{ border: `3px solid ${props.color}` }}>  
      {props.children}  
    </div>  
  );  
}
```

src/components/FancyBorder.js

```
import FancyBorder from "../components/FancyBorder";  
  
function App() {  
  return (  
    <div className="App">  
      <FancyBorder color="blue">  
        <h1>Helle, props.children</h1>  
        <p>It's so convinient tools for us</p>  
      </FancyBorder>  
    </div>  
  );  
}
```

src/App.js



JSX 문법의 장점을 활용!

- 요소 또는 컴포넌트를 전달하고 싶을 때, 반드시 `props.children` 을 사용할 필요는 없습니다
- JSX 문법을 활용하여 Props 의 값에 컴포넌트 또는, 요소를 그려서 전달이 가능합니다!



```
1 function SplitPane(props) {
2   return (
3     <div className="SplitPane">
4       <div className="SplitPane-left">
5         {props.left}
6       </div>
7       <div className="SplitPane-right">
8         {props.right}
9       </div>
10    </div>
11  );
12 }
13
14 function App(props) {
15   return (
16     <SplitPane
17       left={
18         <Contacts />
19       }
20       right={
21         <Chat />
22       }
23     />
24   );
25 }
```



Containment

Containment (방지 X, 담다-포함하다 O)



- Props 또는 Props.children 을 통해 우리는 컴포넌트 또는 요소를 전달하는 방법을 배웠습니다!
- 그럼 이번에는 이전에 만들어 봤던 `<WelcomeDialog>` 를 `<FancyBorder>` 에 담아서 표현해 봅시다!
- 이미 다 만들어 졌으므로 App.js 에서 간단하게 구현이 가능합니다!



```
import WelcomeDialog from "../components/WelcomeDialog";
import FancyBorder from "../components/FancyBorder";

function App() {
  return (
    <div className="App">
      <FancyBorder color="blue">
        <WelcomeDialog />
      </FancyBorder>
    </div>
  );
}
```

src/App.js

Welcome to summoner's lift

소환사의 협곡에 오신 걸 환영합니다.

Containment (방지 X, 담다-포함하다 O)



- 빵이란 컴포넌트 사이에 양파, 양상추, 소스, 불고기 패티를 넣으면? → 불고기 버거
- 빵이란 컴포넌트 사이에 양파, 양상추, 소스, 새우 패티를 넣으면? → 새우 버거
- 즉, 다양한 활용이 가능해 집니다!!



Specialization + Containment



그럼 둘 다 쓰는건 안되나?

당연히 가능!!





Specialization + Containment

- 그럼 이전의 `<Dialog>` 를 사용해서 특수화와 포함을 둘 다 사용해 봅시다!
- 먼저, `<Dialog>` 에 `props.children` 을 추가하여 `Containment` 가 가능하도록 만들어 줍시다!
- 그리고 `<SignUpDialog>` 를 만들어서, 회원 가입 안내 문구(특수화)와 회원 가입 페이지로 이동하는 A 태그(포함)를 전달하여 봅시다!



```
export default function Dialog(props) {  
  return (  
    <div style={{ backgroundColor: props.color }}>  
      <h1>{props.title}</h1>  
      <p>{props.message}</p>  
      {props.children}  
    </div>  
  );  
}
```

src/components/Dialog.js



```
import Dialog from "../Dialog";

export default function SignUpDialog() {
  return (
    <Dialog
      color="skyblue"
      title="안내"
      message="회원 가입이 필요한 서비스 입니다"
    >
      <a href="#">회원 가입 페이지로 이동</a>
    </Dialog>
  );
}
```

src/components/SignUpDialog.js



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

```
function App() {
```

```
  return (
```

```
    <div className="App">
```

```
      <SignUpDialog />
```

```
    </div>
```

```
  );
```

```
}
```

src/App.js

안내

회원 가입이 필요한 서비스 입니다

[회원 가입 페이지로 이동](#)



```
export default function Dialog(props) {  
  return (  
    <div style={{ backgroundColor: props.color }}>  
      <h1>{props.title}</h1>  
      <p>{props.message}</p>  
      {props.children}  
    </div>  
  );  
}
```

src/components/Dialog.js

```
import Dialog from "../Dialog";  
  
export default function SignUpDialog() {  
  return (  
    <Dialog  
      color="skyblue"  
      title="안내"  
      message="회원 가입이 필요한 서비스 입니다"  
    >  
      <a href="#">회원 가입 페이지로 이동</a>  
    </Dialog>  
  );  
}
```

src/components/SignUpDialog.js



FancyBorder 에 넣는건?





```
import FancyBorder from "../components/FancyBorder";
import SignUpDialog from "../components/SignUpDialog";

function App() {
  return (
    <div className="App">
      <FancyBorder color="red">
        <SignUpDialog />
      </FancyBorder>
    </div>
  );
}
```

src/App.js

안내

회원 가입이 필요한 서비스 입니다

[회원 가입 페이지로 이동](#)



조건에 따른 처리



조건에 따른 구현 변경하기!

- 합성(Composition)시 전달되는 props 의 값에 따라서 각기 다른 형태로 출력 되도록 구현도 가능합니다!
- 예를 들어서 지금은 title 의 내용만 props 로 전달이 가능했지만, title로 다른 요소를 전달하여 상황에 따라 컴포넌트를 변화 시켜 봅시다!



Props.title 처리

```
export default function Dialog(props) {  
  return (  
    <div style={{ backgroundColor: props.color }}>  
      {typeof props.title === "string" ? <h1>{props.title}</h1> : props.title}  
      <p>{props.message}</p>  
      {props.children}  
    </div>  
  );  
}
```

src/components/Dialog.js

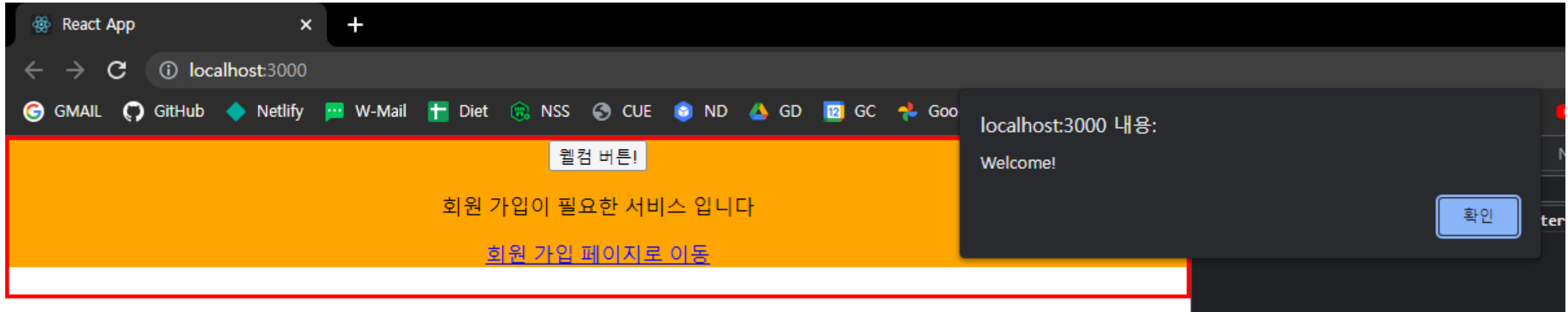
- 전달 된 props.title 이 문자열이면 <h1> 태그 내부에 넣어서 출력하고 아니면 직접 출력하는 형태로 처리



App.js 에서 title로 버튼 요소 전달!

```
function App() {  
  return (  
    <div className="App">  
      <FancyBorder color="red">  
        <Dialog  
          color="orange"  
          title={<button onClick={() => alert("Welcome!")}>웰컴 버튼!</button>}  
          message="회원 가입이 필요한 서비스 입니다"  
        >  
          <a href="#">회원 가입 페이지로 이동</a>  
        </Dialog>  
      </FancyBorder>  
    </div>  
  );  
}
```

src/App.js





실습, 조건에 따른 처리!

- <Dialog> 컴포넌트에 props 로 색상 값이 들어오지 않은 경우에는 기본적으로 orange 색을 색상 값이 들어온 경우에는 해당 색상 값이 적용되도록 컴포넌트를 변경해 봅시다
- 색상 값이 안들어 왔다면, “색상 값이 안들어 왔습니다!” 라는 alert 창도 띄워 줍시다!

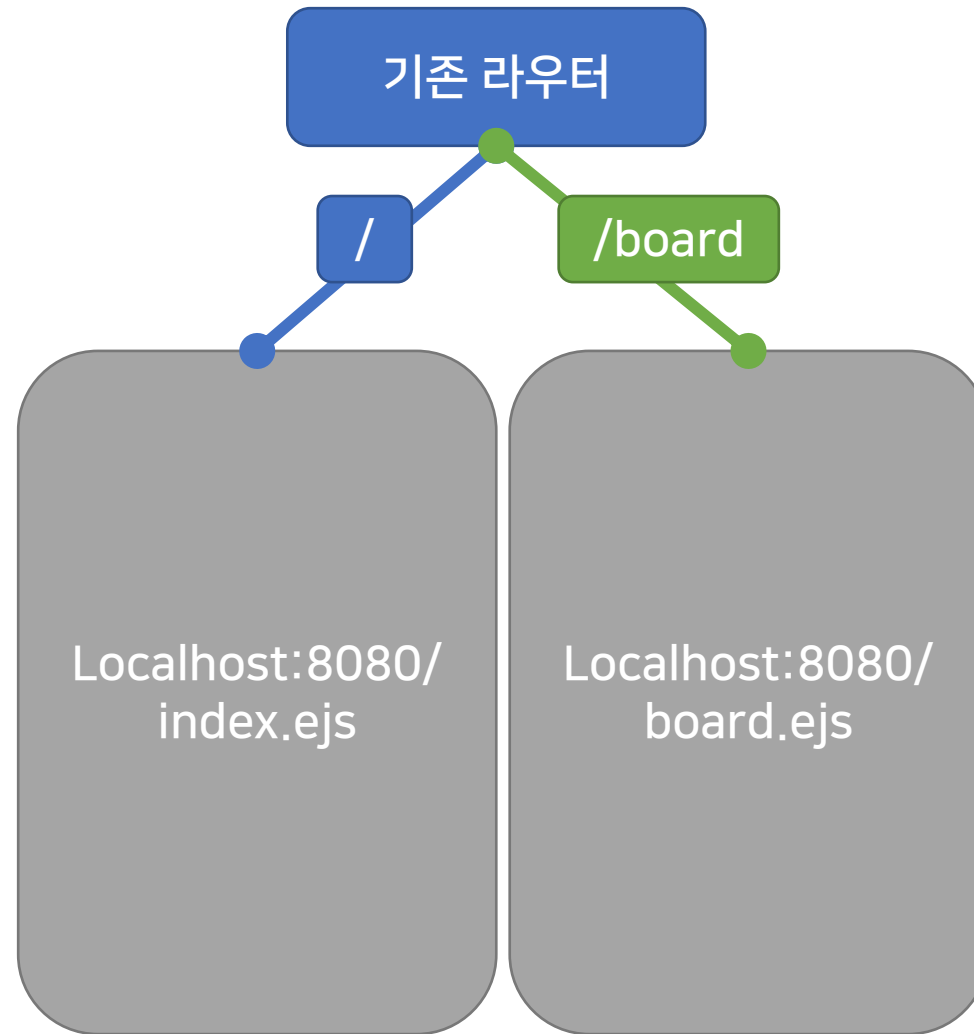


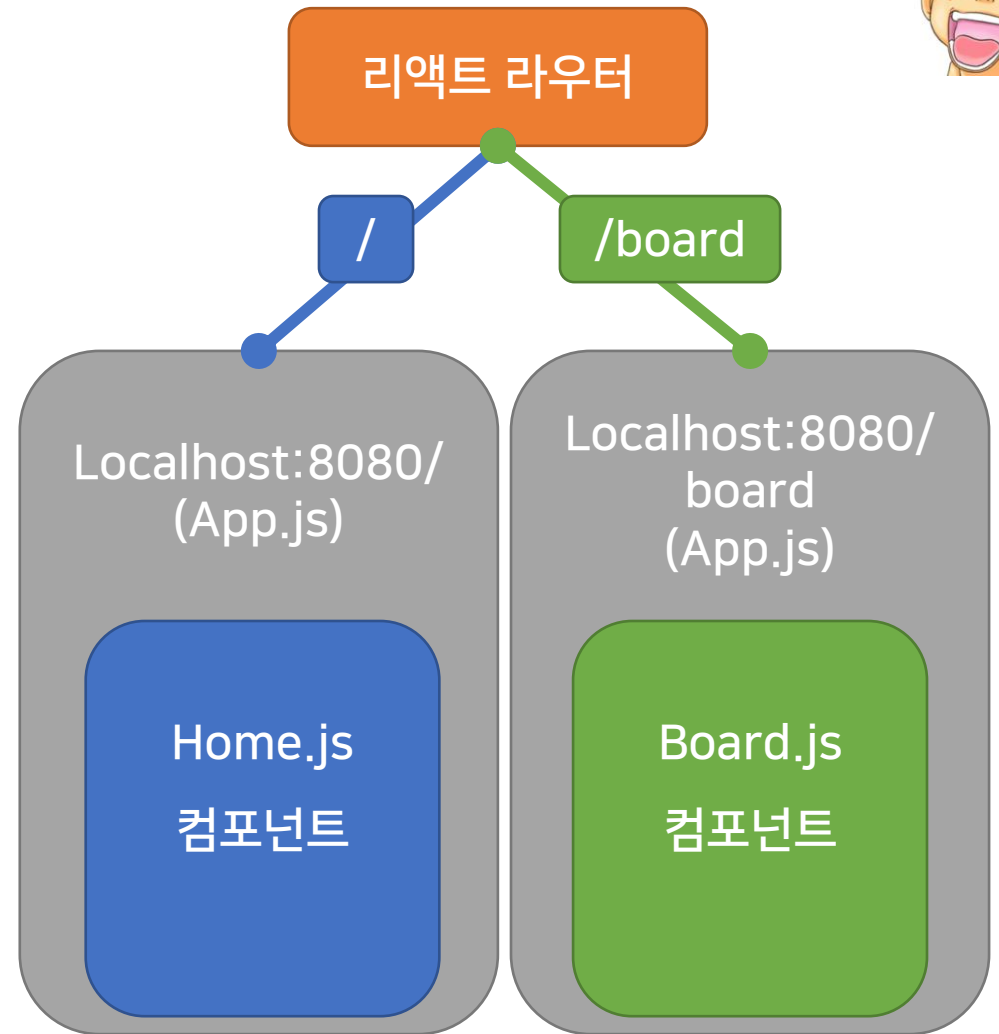
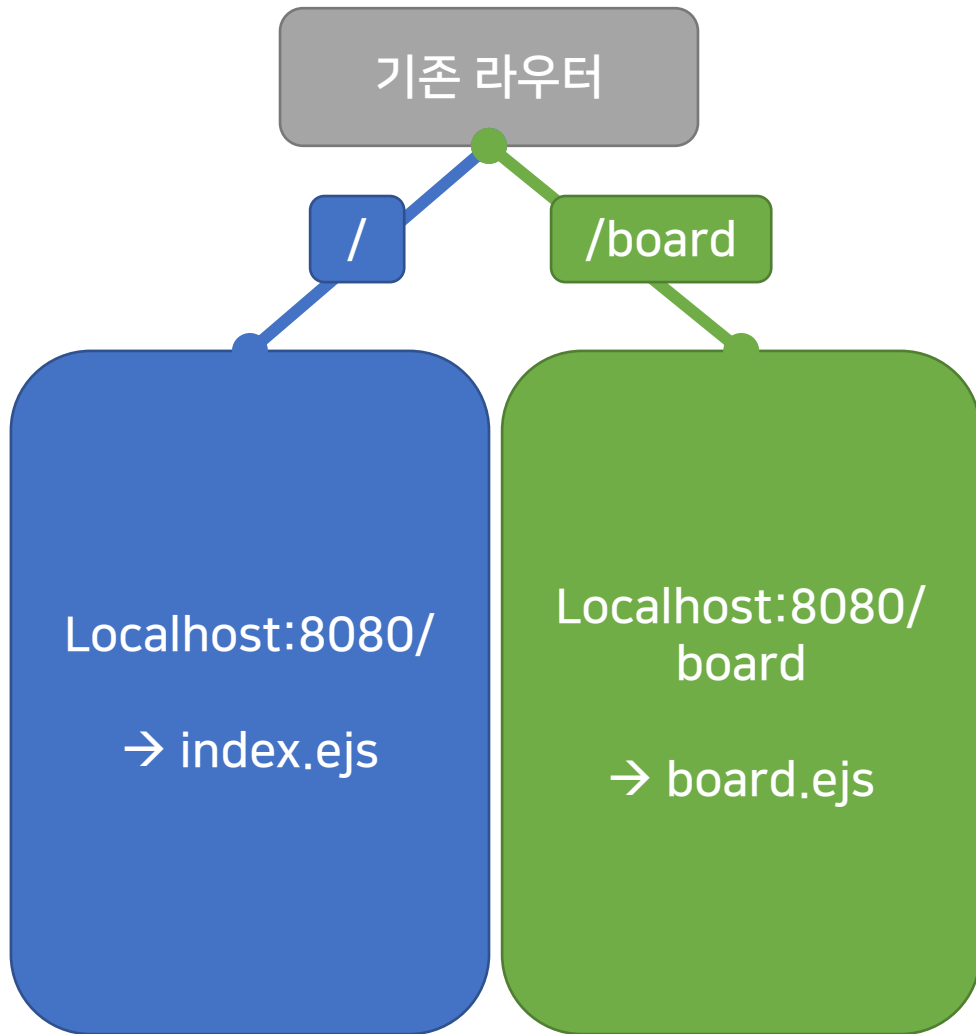
React Router



React 에서 Router 활용하기

- 지금까지 Router 는 입력받은 주소에 따라 페이지를 변경해주는 역할로 많이 사용이 되었습니다.
- 하지만, React 에서는 컴포넌트별(DOM) 라우팅이 가능합니다!
- 조건부 렌더링으로 처리가 가능하지만 서비스의 경우 주소에 따른 구분을 해주어야만 서비스별 구분이 가능하므로 라우팅 기능을 활용이 필요 합니다!
- 그리고 해당 모듈을 쓰면 페이지 깜박임 없이 부드러운 브라우징 가능







React Router 모듈 설치 및 적용

- Npm install react-router-dom
- 그리고 index.js 컴포넌트로 가서 App 컴포넌트를 `<BrowserRouter>` 라는 react-router-dom 이라는 컴포넌트로 감싸 주세요!
- `<BrowserRouter>` 로 감싸 주어야만 `<App>` 컴포넌트에서 발생하는 주소 값의 변경을 감지 할 수 있습니다
- 그외의 라우터로는 `<HashRouter>` 가 유명하며 주소의 해시 주소 `localhost:3000/#hash` 를 감지할 수 있는 라우터 입니다!



```
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </>
);

reportWebVitals();
```

Src/index.js



React Router 용 컴포넌트 만들기

- 라우팅 구현을 위해 h1 태그로만 구성 된 간단한 컴포넌트 2개(Board, Profile)를 만들어 봅시다!



```
export default function Profile() {  
  return (  
    <>  
    <h1>프로필 페이지 입니다</h1>  
    </>  
  )  
}
```

Src/component/Profile.js

```
export default function Board() {  
  return (  
    <>  
    <h1>게시판 페이지 입니다</h1>  
    </>  
  )  
}
```

Src/component/Board.js

App.js 에 Nav 메뉴 구성



- 간단하게 페이지 이동이 가능한 Nav 메뉴도 구현해 봅시다!



```
function App() {  
  return (  
    <div className="App">  
      <nav>  
        <ul>  
          <li>  
            <a href="/profile">프로필 페이지 이동</a>  
          </li>  
          <li>  
            <a href="/board">게시판 페이지 이동</a>  
          </li>  
        </ul>  
      </nav>  
    </div>  
    <Profile />  
    <Board />  
  );  
}
```

```
export default App;
```

Src/App.js



홈 페이지 이동
프로필 페이지 이동
게시판 페이지 이동

프로필 페이지 입니다

게시판 페이지 입니다

- 지금은 라우팅 구현이 안되어 모든 컴포넌트가 하나의 페이지에서 보이고 있습니다! → 이를 해결하기 위해 라우팅을 구현!



React Router 구현

- React-router-dom 의 Route 모듈을 App.js 에 추가하여 컴포넌트 라우팅을 구현해 봅시다!

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



React Router 구현

- React Router 는 Routes 컴포넌트 내부에 Route 컴포넌트를 넣어주고 각각의 주소 값은 path 속성에, 호출할 컴포넌트는 element 속성으로 불러 주면 됩니다!

```
<Routes>
  <Route path="/profile" element={<Profile />} />
  <Route path="/board" element={<Board />} />
</Routes>
```

Src/App.js

- 여기서 Route 는 반드시 Routes 내부에 있어야 합니다!



React Router 구현

- `<a>` 태그는 브라우저 레벨에서 페이지를 자동으로 새로고침 하기 때문에 React 에서는 이를 막고자 `<Link to="">` 라는 컴포넌트를 사용합니다!
- Link 컴포넌트는 html 상에서는 `<a>` 태그로 변경이 되지만 브라우저 새로고침 없이 주소만 변경해 주는 역할을 합니다!
- `<a>` 태그를 `<Link>` 컴포넌트로 변경!



```
<nav>
  <ul>
    <li>
      <Link to="/">홈 페이지 이동</Link>
    </li>
    <li>
      <Link to="/profile">프로필 페이지 이동</Link>
    </li>
    <li>
      <Link to="/board">게시판 페이지 이동</Link>
    </li>
  </ul>
</nav>
```

Src/App.js



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

```
function App() {
  return (
    <div className="App">
      <nav>
        <ul>
          <li>
            <Link to="/">홈 페이지 이동</Link>
          </li>
          <li>
            <Link to="/profile">프로필 페이지 이동</Link>
          </li>
          <li>
            <Link to="/board">게시판 페이지 이동</Link>
          </li>
        </ul>
      </nav>
      <Routes>
        <Route path="/profile" element={<Profile />} />
        <Route path="/board" element={<Board />} />
      </Routes>
    </div>
  );
}
```

```
export default App;
```

전체 코드

Src/App.js



Localhost:3000/



[홈 페이지 이동](#)
[프로필 페이지 이동](#)
[게시판 페이지 이동](#)

Localhost:3000/profile



[홈 페이지 이동](#)
[프로필 페이지 이동](#)
[게시판 페이지 이동](#)

프로필 페이지 입니다

Localhost:3000/board



[홈 페이지 이동](#)
[프로필 페이지 이동](#)
[게시판 페이지 이동](#)

게시판 페이지 입니다



React Router

심화 활용



React Router 심화 활용

- 이번에는 `<Header>` 컴포넌트를 만들고 해당 `<Header>` 를 통한 라우팅 처리를 구현해 봅시다!
- 그 외에도 주소 예외 처리 및 주소의 parameter 사용에 대해서도 배워 봅시다!



Header 컴포넌트 작성

- 부드러운 브라우징을 위해 `<Link>` 컴포넌트 사용
- Header 에 맞게 `display: "flex"` 처리



```
import { Link } from 'react-router-dom'

export default function Header() {
  return (
    <>
      <nav>
        <ul style={{ display: "flex", justifyContent: "space-around" }}>
          <li>
            <Link to="/">홈 페이지 이동</Link>
          </li>
          <li>
            <Link to="/profile">프로필 페이지 이동</Link>
          </li>
          <li>
            <Link to="/board">게시판 페이지 이동</Link>
          </li>
        </ul>
      </nav>
    </>
  )
}
```

Src/component/Header.js



App.js 에서의 라우팅 처리

- 그럼 **<Header>** 컴포넌트에서 작성한 주소에 맞게 라우팅 처리를 하고 각각의 컴포넌트를 제작해 봅시다!
- 메인 페이지는 **<Header>** 컴포넌트만 불러오면 되므로 **"/"** 주소에는 **<Header>** 컴포넌트만 부여
- 각각의 주소는 각각 주소에 맞는 컴포넌트 부여



```
import { Route, Routes } from "react-router-dom";
import Profile from "../components/Profile";
import Board from "../components/Board";
import Header from "../components/Header";
import NotFound from "../components/NotFound";

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Header />} />
        <Route path="profile" element={<Profile />} />
        <Route path="board" element={<Board />} />
      </Routes>
    </div>
  );
}

export default App;
```

Src/App.js



Profile, Board 컴포넌트 업데이트!

- 이제는 App.js 에서 컴포넌트를 부르는 방식이 아니라 해당 컴포넌트 자체가 그려지는 구조를 가지므로 각각 컴포넌트에도 `<Header>` 컴포넌트 추가 필요!



```
import Header from './Header'

export default function Profile() {
  return (
    <>
      <Header />
      <h1>프로필 페이지 입니다</h1>
    </>
  )
}
```

Src/component/Profile.js

```
import Header from './Header'

export default function Board() {
  return (
    <>
      <Header />
      <h1>게시판 페이지 입니다</h1>
    </>
  )
}
```

Src/component/Board.js

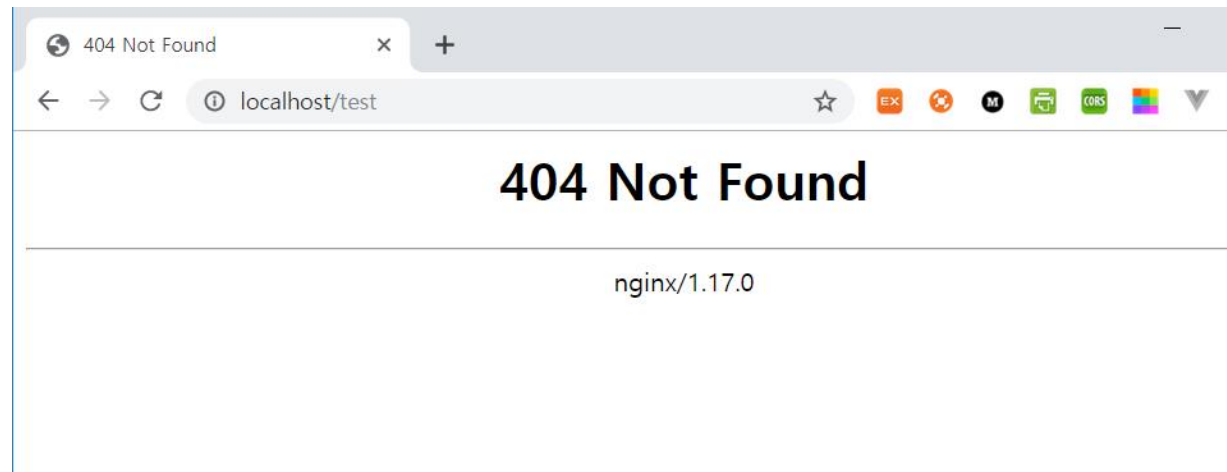


주소 예외 처리



Page Not Found

- 실제 서비스의 경우 사용자가 예상하지 못한 주소 값을 입력하는 경우가 발생 합니다.
- 이럴 때, 브라우저에서 제공하는 **404 Not Found** 페이지를 띄우면 일단 서버 응답을 기다리는 시간도 오래 걸릴 뿐더러, 결과 페이지가 서비스의 신뢰를 깨는 역할을 하게 됩니다!





Page Not Found

- 따라서 잘못 입력 된 주소에 대한 예외 처리가 필요합니다!
- React-router-dom 모듈은 해당 부분에 있어서 * 라는 편리한 방법을 제공합니다!
- * 는 모든 주소 입력을 의미하며 아래와 같이 사용합니다!

```
<Route path="*" element={<NotFound />} />
```



```
function App() {  
  return (  
    <div className="App">  
      <Routes>  
        <Route path="/" element={<Header />} />  
        <Route path="profile" element={<Profile />} />  
        <Route path="board/" element={<Board />} />  
        <Route path="*" element={<NotFound />} />  
      </Routes>  
    </div>  
  );  
}  
  
export default App;
```

코드 처리 방향

Src/App.js



Page Not Found

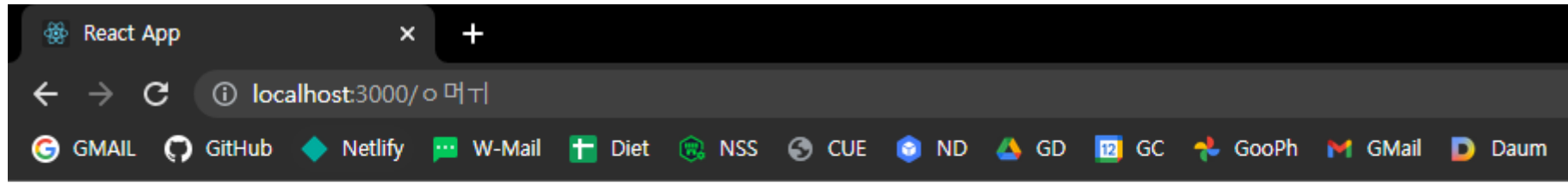
- React-router-dom 라우터도 백엔드 라우터와 마찬가지로 코드 선언 순서에 따라 처리가 됩니다!
- 위에서 주소 처리를 해도 일치가 되는 부분이 없으면 아래의 라우터로 내려오게 되는데 마지막 라우터에서 주소를 * 를 사용하여 처리 하면 일치가 안된 주소는 한꺼번에 처리가 가능합니다!
- 해당 라우터에서 <NotFound> 컴포넌트로 연결하여 모든 예외를 처리!



```
import Header from './Header'

export default function NotFound() {
  return (
    <>
      <Header />
      <h1>Page Not Found</h1>
    </>
  )
}
```

Src/component/NotFound.js



- [홈 페이지 이동](#)

- [프로필 페이지 이동](#)

- [게시판 페이지 이동](#)

Page NotFound



주소

Parameter 활용



주소의 Parameter 값 활용

- 백엔드에서 주소로 전달되는 Parameter 와 Query 는 중요하게 사용이 됩니다!
- 물론 리액트 라우터에서도 두 가지 모두를 사용할 수 있습니다!
- <Board> 컴포넌트에 2개의 게시글이 있다고 가정하여 parameter 를 활용하여 봅시다!



Board 컴포넌트 변경

- <Board> 컴포넌트는 이제 2개의 게시글의 목록을 보여주고 해당 게시글로 이동하는 역할을 합니다!

```
import { Link, Route, Routes } from 'react-router-dom';
import Header from './Header'

export default function Board() {
  return (
    <>
      <Header />
      <h1>게시판 페이지 입니다</h1>
      <Link to="1"><h2>게시글 1번 보여주기</h2></Link>
      <Link to="2"><h2>게시글 2번 보여주기</h2></Link>
    </>
  )
}
```

Src/component/Board.js



주소 처리 방법, to??

- 어떤 주소는 / 로 시작하고 어떤 주소는 / 가 없이 시작하죠?
- / 로 시작
 - 앱의 기본 주소인 `Localhost:3000` 뒤에 / 뒤의 주소가 이어짐
 - `/profile` → `Localhost:3000/profile`



주소 처리 방법, to??

- / 없이 시작
 - 현재 라우팅 된 주소의 뒤에 해당 주소의 문자열이 추가 됨
 - 현재 라우팅 주소가 localhost:3000/board 일 때
 - 1 → localhost:3000/board/1



Route 에 Parameter 선언

- App.js 에 선언 된 라우터 선언부에 Parameter 를 선언해 봅시다!
- 기존과 같은 방법으로 주소/:parameter 로 선언하면 됩니다!

```
<Routes>
  <Route path="/" element={<Header />} />
  <Route path="profile" element={<Profile />} />
  <Route path="board" element={<Board />} />
  <Route path="board/:boardID" element={<BoardDetail />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```

Src/App.js

BoardDetail 컴포넌트에서 parameter 받기



- 게시글 내용은 <BoardDetail> 컴포넌트에서 받으므로 해당 컴포넌트를 만들어 줍시다!
- Parameter 로 전달 받은 값도 받아서 활용해 줍시다!
- Parameter 는 useParams 로 받을 수 있으며, useParams 로 선언한 객체 변수에 담기게 됩니다
- parameter 로 선언한 이름이 Key 로 설정 됩니다!



```
import { useParams } from "react-router-dom"
import Header from "../Header";
```

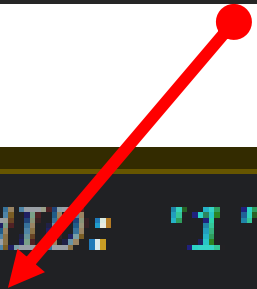
```
export default function BoardDetail() {
  const params = useParams();
  console.log(params);
  return (
    <>
      <Header />
      <h2>{params.boardID} 번 게시글 내용입니다!</h2>
    </>
  )
}
```

Src/component/BoardDetail.js



전달한 이름 그대로 객체의 Key 값이 배정

```
<Route path="board/:boardID" element={<BoardDetail />} />
```

A red arrow originates from the `boardID` property in the `path` attribute of the `<Route>` component in the code block above. It points diagonally down and to the left towards the `boardID` property in the object shown in the console log below.

```
▼ {boardID: '1'} ⓘ  
  boardID: "1"  
  ► [[Prototype]]: Object
```



구조 분해 할당 문법!? 가능!!

```
import { useParams } from "react-router-dom"
import Header from "../Header";

export default function BoardDetail() {
  const { boardID } = useParams();
  return (
    <>
      <Header />
      <h2>{boardID} 번 게시물 내용입니다!</h2>
    </>
  )
}
```

Src/component/BoardDetail.js



주소

Query 받기



주소의 Query 받기

- 하나하나 parameter 로 받기 어려운 값들의 경우는 주소 뒤에 ? 를 붙이고 시작하는 Query 로 받았었습니다!
- <http://localhost:3000/board/1?title=title&content=lorem>
- 원래 Query 는 react-query 모듈을 사용해서 받습니다.
- Query 를 제대로 사용하는 경우에는 react-query 모듈이 제공하는 다양한 기능을 활용하는 방법이 좋지만 이번에는 간단하게 Query가 받아지는지 확인만 할 것이므로 react-router-dom 의 기능을 활용해 보겠습니다!



주소의 Query 받기

- React-router-dom 의 `useLocation` 을 사용하면 아래의 것을 받을 수 있습니다
 - Search: ? 를 포함한 쿼리 스트링
 - Path: 현재의 주소
 - Hash: 주소에 들어있는 # 문자열 뒤의 값(해시 주소 값)



모듈 호출 및 적용하기

```
import { useLocation, useParams } from "react-router-dom";
import Header from "../Header";

export default function BoardDetail() {
  const { boardID } = useParams();
  const location = useLocation();
  return (
    <>
      <Header />
      <h2>{boardID} 번 게시글 내용입니다!</h2>
    </>
  );
}
```

Src/component/BoardDetail.js



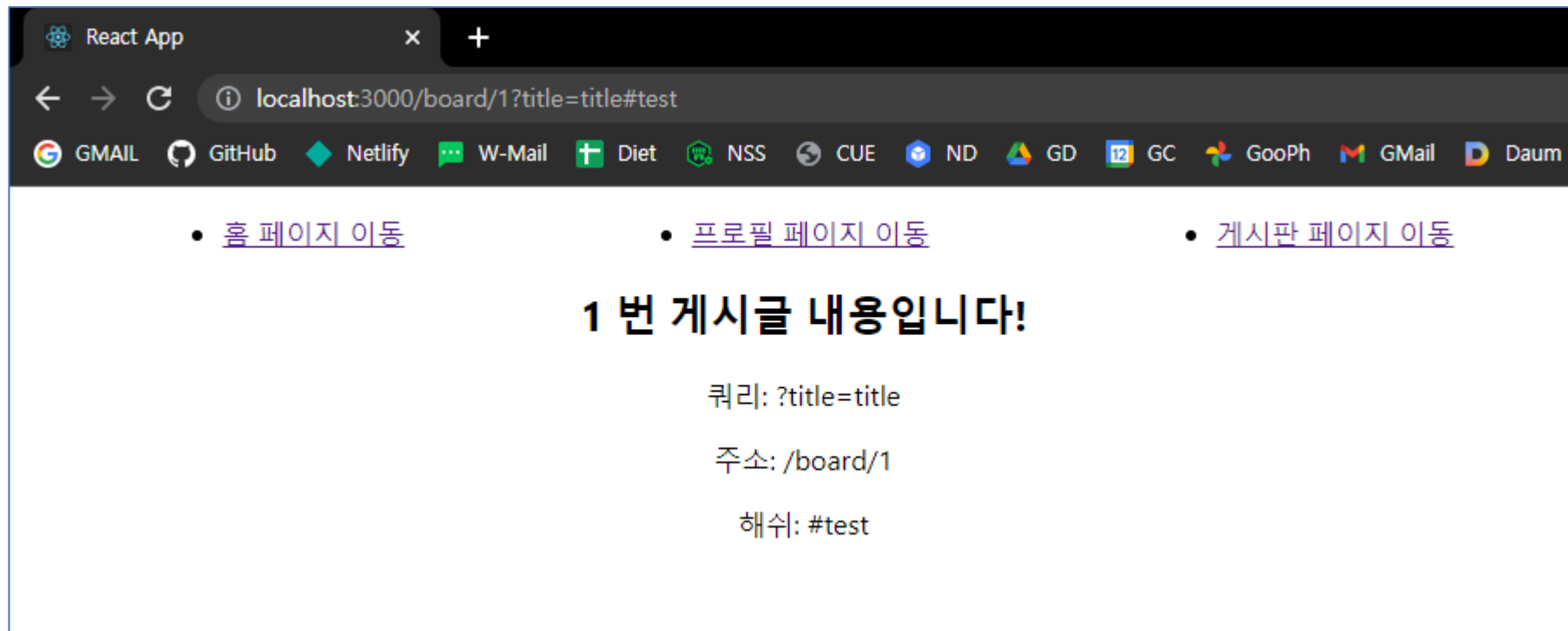
useLocation 값 보기

```
import { useLocation, useParams } from "react-router-dom";
import Header from "../Header";

export default function BoardDetail() {
  const { boardID } = useParams();
  const location = useLocation();
  return (
    <>
      <Header />
      <h2>{boardID} 번 게시물 내용입니다!</h2>
      <p>쿼리: {location.search}</p>
      <p>주소: {location.pathname}</p>
      <p>해쉬: {location.hash}</p>
    </>
  );
}
```

Src/component/BoardDetail.js

<http://localhost:3000/board/1?title=title#test>





수고하셨습니다!