

React

chapter01

리액트 기초

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

part.1

리액트 소개

part.2

실습 환경 구축

part.3

소스 코드 수정 방법

part.4

JSX

part.5

컴포넌트 만들기

part.6

props

part.7

이벤트

part.8

state

part.9

Create

part.10

Update

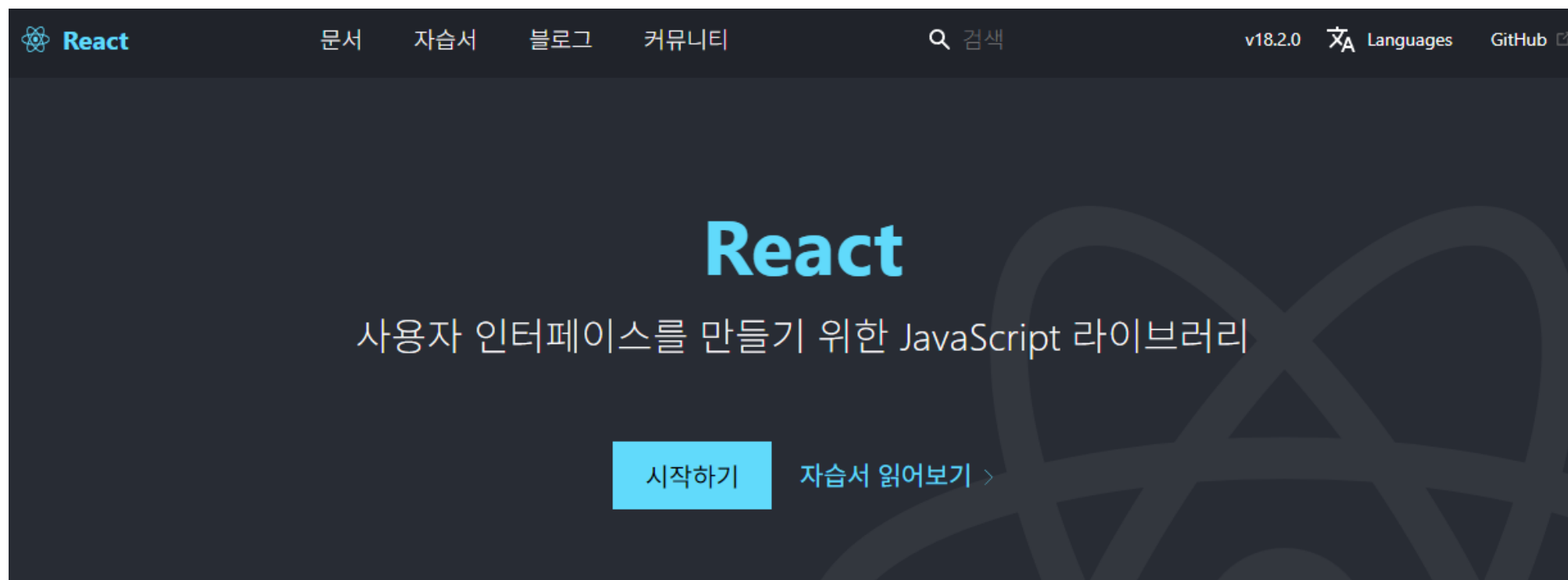
part.11

Delete

리액트 소개

리액트란?

리액트란 사용자 인터페이스(버튼, 입력필드 등)를 효율적으로 만들기 위한 자바스크립트 라이브러리이다.
이와 유사한 자바스크립트 라이브러리에는 Vue.js, Angular 등이 있다.



<https://ko.legacy.reactjs.org>

웹 페이지 구조를 구성하는 방식은 MPA와 SPA가 있다.

- **MPA(Multi-Page Application)**

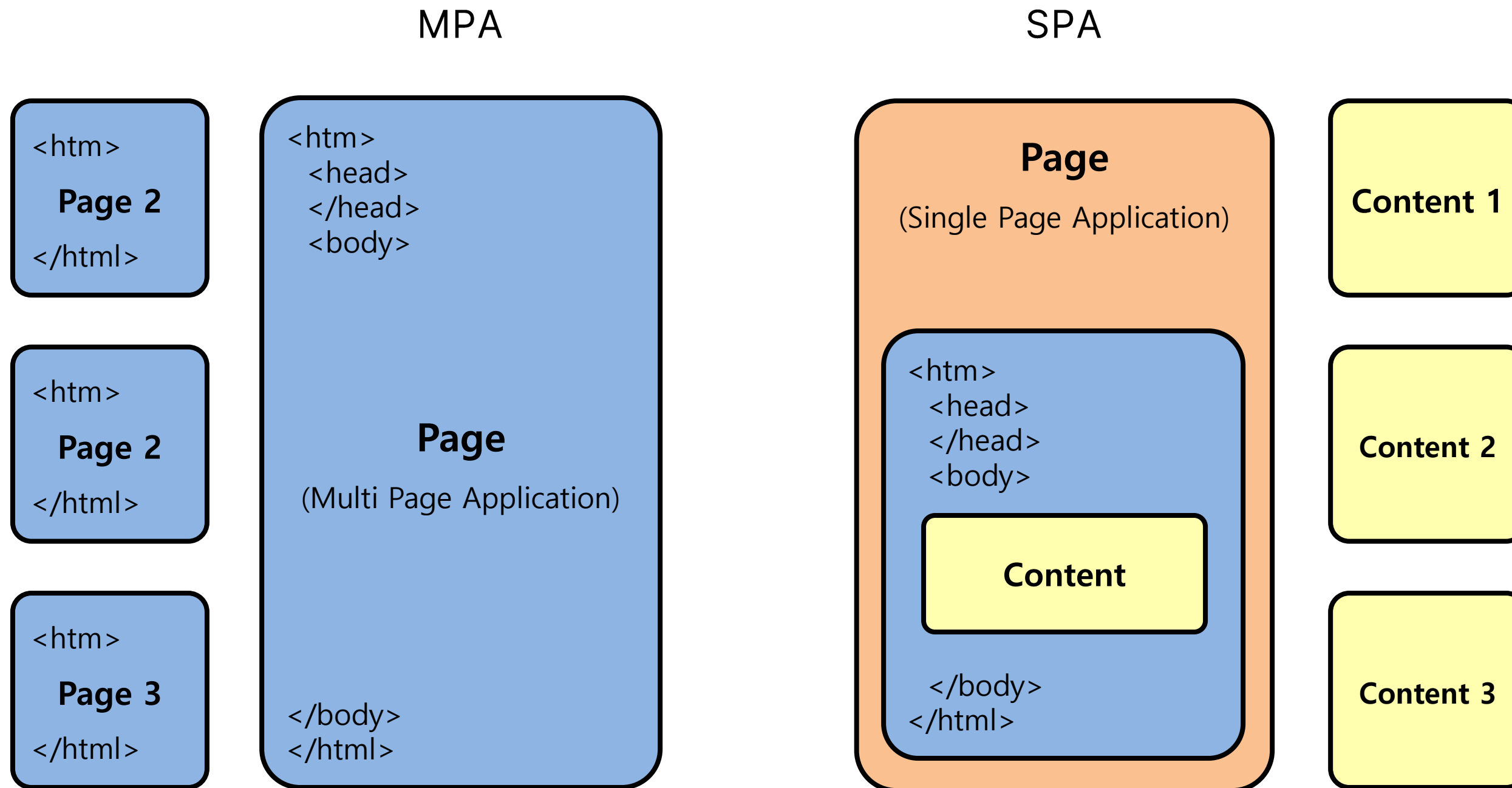
각 페이지마다 별도의 HTML 파일이 존재하며, 사용자가 페이지를 이동할 때마다 해당 페이지의 HTML 파일을 서버에서 받아와서 렌더링하는 방식이다.

- **SPA(Single Page Application)**

하나의 HTML 페이지로 동작하는 웹 사이트 방식이다.

규모가 큰 사이트는 수십~수백개의 페이지를 각각 만드는 것이 비효율적이다.

그래서 기본 HTML 틀을 두고 사용자가 특정 페이지를 요청할 때 필요한 데이터만 서버에서 받아와서 동적으로 채워 넣는다. React는 SPA 방식을 사용하여 페이지 이동 시 필요한 부분만 업데이트 한다.



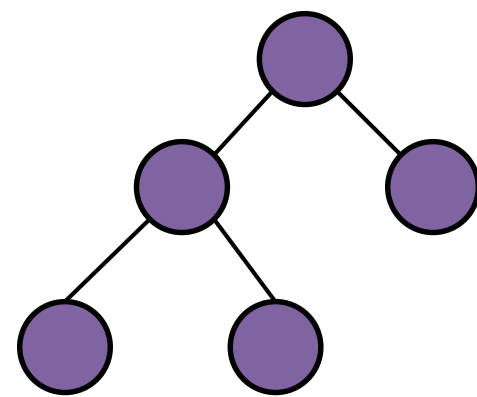
1. 빠른 렌더링 속도

React는 화면을 빠르게 렌더링하기 위해 내부적으로 Virtual DOM(가상 돔)을 사용한다.

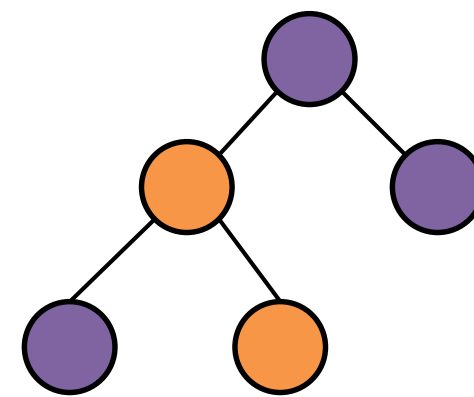
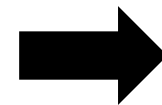
Virtual DOM은 Real DOM의 사본으로 빠르게 동작하여, 웹 페이지와 Real DOM 사이에서 중간 매개체 역할을 한다.

기존의 방식에서는 화면을 업데이트할 때 DOM을 직접 수정해야 했고, 수정할 부분을 찾기 위해 여러 번 연산해야 해서 성능 저하가 발생했다.

반면, Virtual DOM 방식은 DOM을 직접 수정하지 않고, 가상 돔을 먼저 만들고 이전 가상 돔과 비교하여 변경할 부분만 찾아서 한번에 업데이트한다.



이전 가상 DOM



변경된 새로운 가상 DOM

2. 코드 재사용

React에서 페이지를 여러 컴포넌트로 구성한다.

컴포넌트는 화면을 구성하는 부품으로, 여러 페이지에서 재사용할 수 있어 코드를 효율적으로 활용할 수 있다.

기존에 개발해 둔 컴포넌트를 재사용함으로써 개발 시간을 줄일 수 있다.

공통으로 사용하는 컴포넌트에 문제가 생기면 해당 컴포넌트만 수정하면 된다.

3. 모바일 앱 개발 가능

React를 알고 있다면 React Native를 쉽게 배울 수 있다.

React Native는 모바일 앱 개발 프레임워크로, 안드로이드와 IOS 앱을 동시에 개발할 수 있다.

Node.js란?

Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임 환경 이다.
브라우저가 아닌 환경에서 JavaScript를 실행할 수 있는 환경을 제공한다.
JavaScript는 프로그래밍 언어이고, Node는 JavaScript 코드를 실행하는 도구이다.



Node.js의 특징

JavaScript는 클라이언트 언어지만, Node를 사용하면 서버 기능도 구현할 수 있다.
Node를 사용하면 브라우저나 HTML 없이도 JavaScript를 실행할 수 있다.

npm


Node의 패키지 매니저로 프로젝트에서 필요한 외부 패키지들을 설치하고 삭제할 수 있다.
Node를 설치하면 자동으로 npm도 함께 설치된다.

바닐라 자바스크립트 수업에서 이미 Node를 설치했으므로, 여기서는 설치 과정을 생략하겠습니다.
설치가 안된 분은 "바닐라 자바스크립트 - 2장" PPT를 보고 설치해주세요.

Node가 설치되어 있는지 확인해보겠습니다.

CMD를 열고 다음 명령어를 입력하세요. "node -v"

Node의 버전이 출력되면 설치가 완료된 것입니다.



A screenshot of a Windows Command Prompt window. The title bar reads '명령 프롬프트' (Command Prompt). The window content shows the following text:

```
Microsoft Windows [Version 10.0.19041.1]
(c) Microsoft Corporation. All rights reserved.

C:\Users\G201>node -v
v20.12.2
```

비주얼 스튜디오 코드(Visual Studio Code)란?

VSCode는 Microsoft에서 개발한 소스 코드 에디터로, 다양한 프로그래밍 언어를 지원한다.

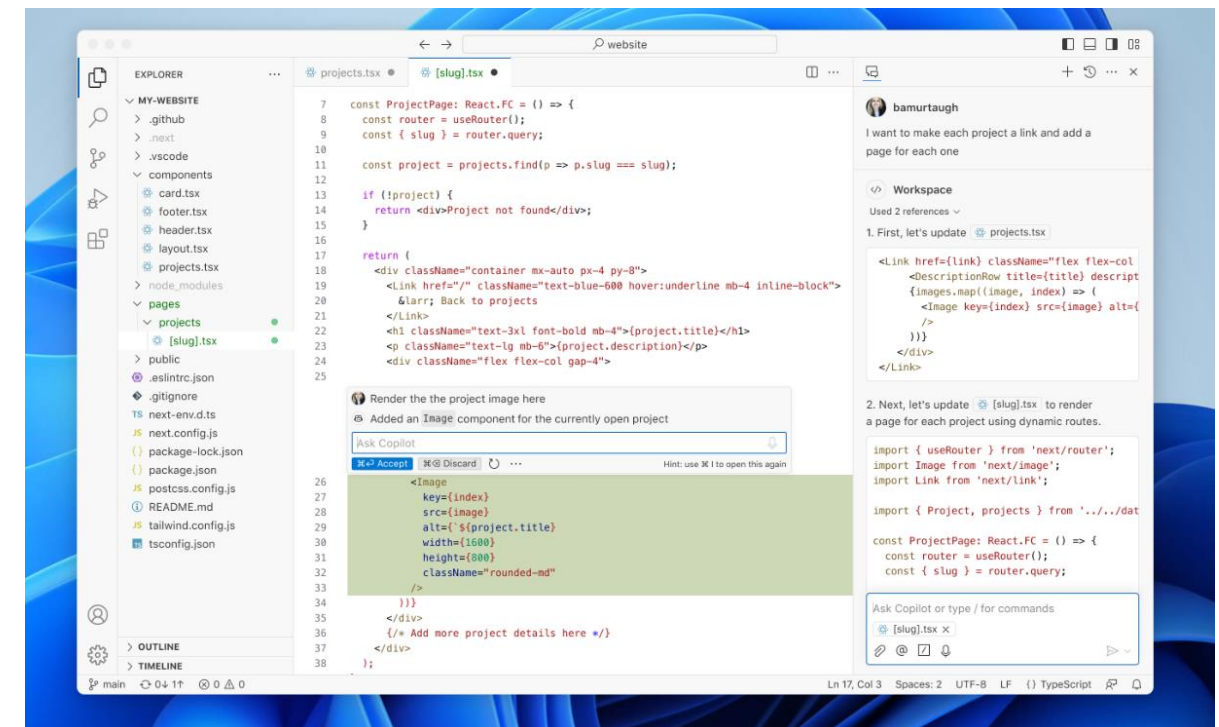
특히 JavaScript, React, Angular 같은 프론트 개발에 최적화 되어 있다.

비주얼 스튜디오 코드 특징

경량 에디터로 다른 에디터에 비해 빠르고 가볍다.

디버깅, Git, 터미널 등 다양한 기능을 제공한다.

마찬가지로 이전 수업에서 이미 VSCode를 설치했으므로,
여기서는 설치 과정을 생략하겠습니다.



실습 환경 구축

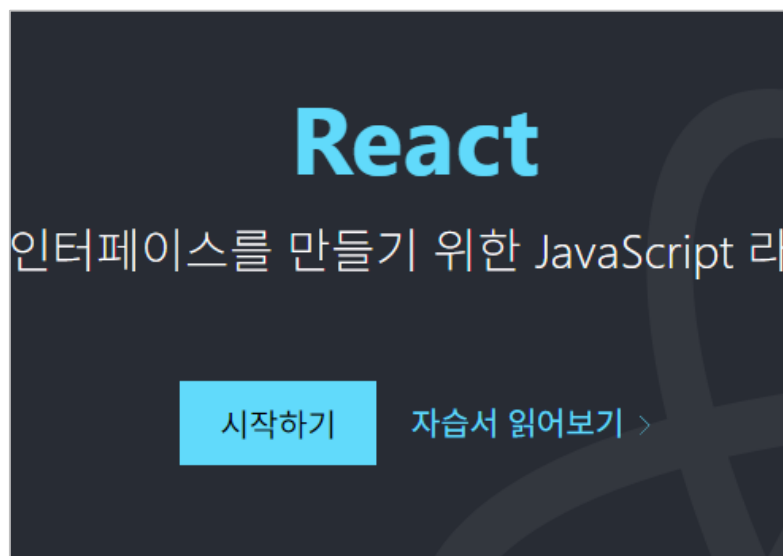
React 프로젝트 생성하기

React 공식 홈페이지에 접속하세요.

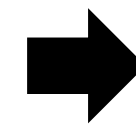
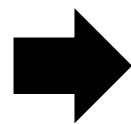
화면 가운데 있는 '시작' 버튼을 클릭하세요.

사이드바에서 '새로운 React 앱 만들기' 메뉴를 선택하세요.


화면 중간에 'Create React App' 부분을 확인하세요.



<https://ko.reactjs.org>



1. 먼저 react-workspace 폴더를 생성하세요.
2. VSCode를 여세요.
3. 터미널을 열고, cd 명령으로 위치를 workspace로 이동하세요.
4. "npx create-react-app react-app" 명령을 입력하여 React 프로젝트를 생성하세요.

 react-workspace

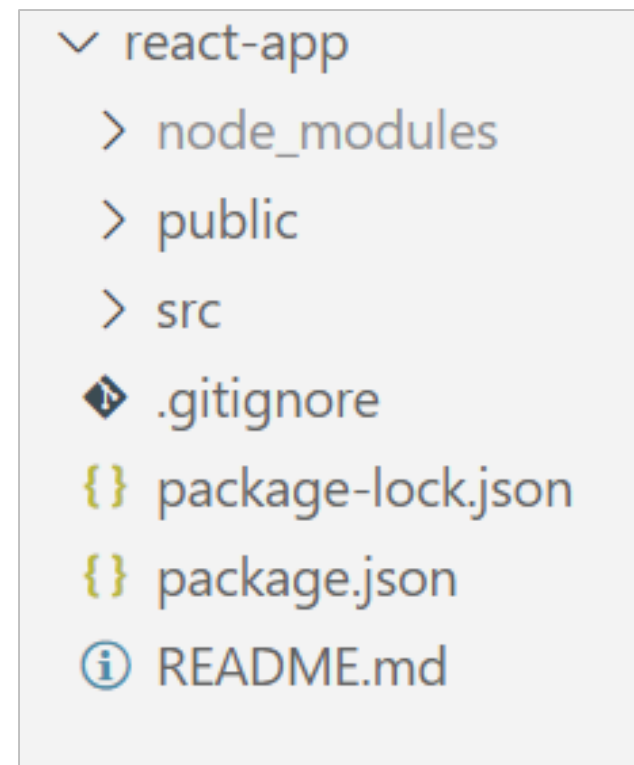
```
문제   출력   디버그 콘솔   터미널   포트
PS C:\Users\imjiyeon> cd D:\0.tool\react-workspace
PS D:\0.tool\react-workspace> npx create-react-app react-app

Creating a new React app in D:\0.tool\react-workspace\react-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

생성한 프로젝트 폴더를 불러온다.

React 구조가 생성된 것을 확인할 수 있다.



콘솔창에서 React 관련 명령어들이 출력된 것을 확인할 수 있다.

npm start 명령어를 입력하면 React 앱이 실행된다.

```
문제  출력 디버그 콘솔 터미널 포트
Success! Created react-app at C:\imjiyeon\__react\
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.
```

You can now view **react-app** in the browser.

Local: `http://localhost:3000`
On Your Network: `http://192.168.0.67:3000`

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

webpack compiled **successfully**

소스 코드 수정 방법 프로젝트 구조

src

index.js: React 어플리케이션의 진입점으로, React 컴포넌트를 렌더링하는 파일

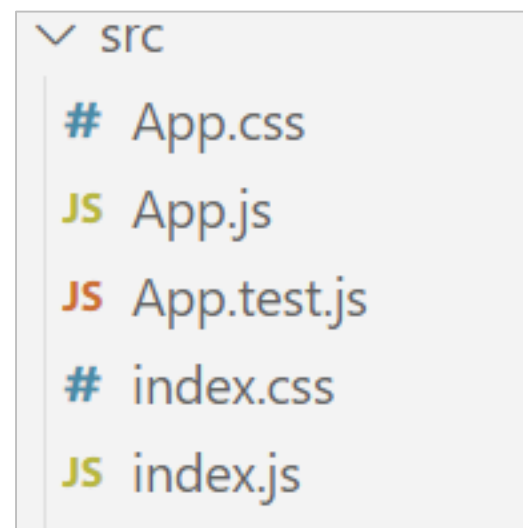
Index.css: 어플리케이션 전체에 적용되는 스타일

App.js: 주요 컴포넌트를 정의하는 파일로, 핵심 UI구조를 담당하는 파일

App.css: 특정 페이지에 적용되는 스타일

public

index.html: 브라우저에 표시되는 기본 HTML 템플릿 파일



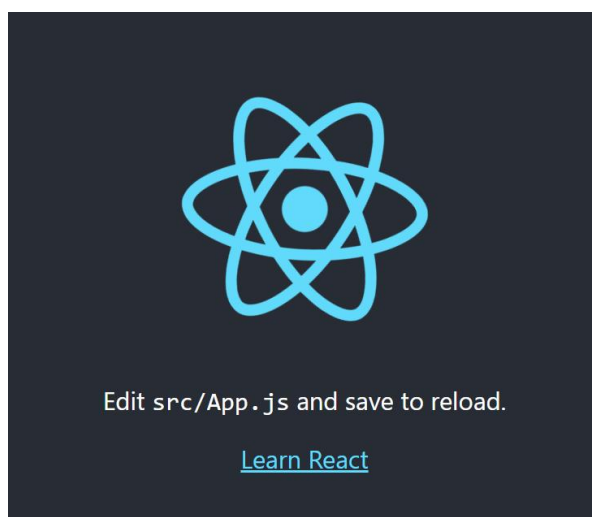
소스 코드 수정 방법 프로젝트 실행

npm start

개발시 어플리케이션을 실행하는 명령이다.

모든 소스코드를 처리하기 때문에 용량이 커서, 실제 배포 환경에서는 적합하지 않다.

실제 서비스를 배포할 때는 프로젝트를 빌드하고, 빌드된 파일을 사용해야 한다.



소스 코드 수정 방법 소스코드 수정

index.js를 보면 App이라는 컴포넌트를 포함하고 있다.
App 태그를 제거하면 화면에 아무것도 나타나지 않는다.
App 이라는 태그가 전체 화면이다.

```
root.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>  
);
```

App.js에서 불필요한 코드를 제거한다.

```
function App() {  
  return (  
    <div className="App">  
      <header className="App-header">  
        <img src={logo} className="App-logo" alt="logo" />  
        <p>  
          Edit <code>src/App.js</code> and save to reload.  
        </p>  
        <a
```

소스 코드 수정 방법

소스코드 수정

<div> 안에 내용을 "Hello React"로 변경한다.
화면에 Hello React가 출력되는 것을 확인한다.
이렇게 App.js를 편집하여 UI를 만들 수 있다.

```
function App() {  
  return (  
    <div className="App">  
      Hello React!  
    </div>  
  )  
}
```

Hello React!

App.css 파일 안에 내용을 모두 지우고, 브라우저를 확인하면 글자가 왼쪽으로 정렬된다.

```
App {  
  text-align: center;  
}  
  
.App-logo {  
  height: 40vmin;  
  pointer-events: none;  
}  
  
@media (prefers-reduced-motion: no-preference) {  
  .App-logo {  
    animation: App-logo-spin infinite 20s linear;  
  }  
}
```

Hello React!

소스 코드 수정 방법 소스코드 수정

App.css 파일에 다음과 같이 스타일을 추가하면, 배경화면이 회색으로 변경된다.

```
body {  
  background-color: gray;  
}
```



Hello React!

Index.css 파일의 내용을 모두 지우면, 화면에서 폰트가 변경되고 여백이 사라진다.

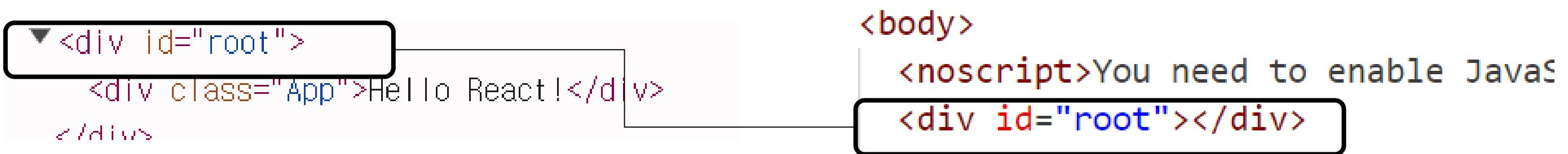
```
body {  
  margin: 0;  
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',  
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',  
    sans-serif;  
  -webkit-font-smoothing: antialiased;  
  -moz-osx-font-smoothing: grayscale;  
}
```

소스 코드 수정 방법 소스코드 수정

개발자 도구에서 Element 탭을 보면, `<div id="root">` 라는 태그가 있다.

index.html 파일을 보면 여기에 id가 root인 태그가 있다.

App.js의 내용이 root 태그 아래 추가된다는 것을 알 수 있다.



root 태그에 style을 추가하면 화면 전체에 빨간색 테두리가 추가된다.

```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root" style="border: 10px solid red;"></div>
```

Hello React!

소스 코드 수정 방법 프로젝트 빌드

npm run build

프로젝트를 패키지 파일로 빌드하는 명령이다.

build 명령을 실행하면 build라는 폴더가 생성된다.

생성된 index.html 파일을 열어보면 공백이 하나도 없는 것을 확인할 수 있다.

이는 파일의 용량을 줄이기 위해 불필요한 요소를 최대한 제거한 것이다.

빌드 -> npm 설치 -> 서비스 실행

```
문제   출력   디버그 콘솔   터미널   포트
C:\imjiyeon\__react\workspace\react-app> npm run build
> react-app@0.1.0 build
> react-scripts build
```

```
npm install -g serve
```

```
serve -s build
```

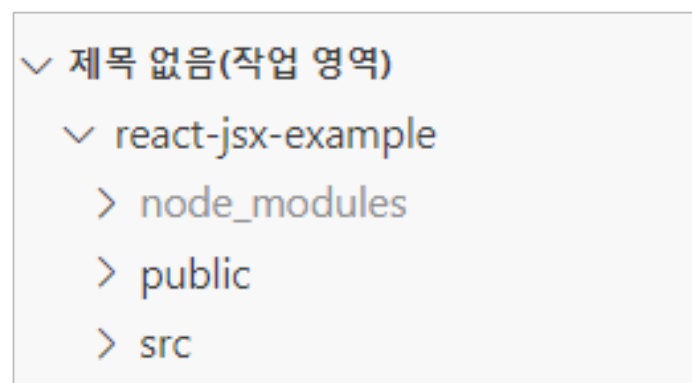
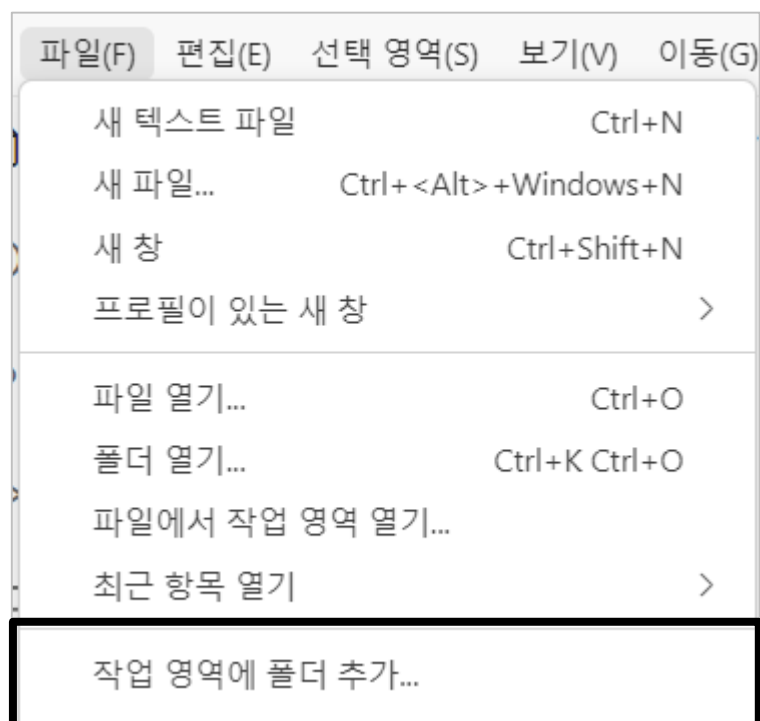
```
▼ react-app
  ▼ build
    > static
    {} asset-manifest.json
    ★ favicon.ico
    <> index.html

<> index.html ×
react-app > build > <> index.html > html > head > script
1 <!doctype html><html lang="en"><head><meta charse
><meta name="viewport" content="width=device-widt
content="#000000"/><meta name="description" conte
rel="apple-touch-icon" href="/logo192.png"/><link
```

“react-jsx-example” 이라는 이름으로 새로운 React 프로젝트를 생성한다.

```
npx create-react-app react-jsx-example
```

생성한 프로젝트 폴더를 VSCode에 불러온다.



JSX는 JavaScript와 XML/HTML을 합친 문법으로, JavaScript XML의 앞글자를 따서 JSX라고 한다.
JSX 자바스크립트를 확장한 문법으로 HTML를 자바스크립트 코드 안에서 사용할 수 있도록 도와준다.
JSX를 사용하면 HTML 문법으로 UI를 생성할 수 있다.

자바스크립트 변수에 HTML 코드 대입

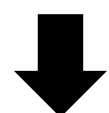
```
let element = <div>Hello</div>
```

JSX 문법으로 작성한 코드는 자바스크립트 코드로 변환된다.

createElement라는 함수로 변경되며, 이 함수에 의해 엘리먼트(Element)가 생성된다.

엘리먼트는 UI를 구성하는 자바스크립트 객체이다.

```
let element = <div>Hello</div>
```



```
React.createElement("div", null, "Hello");
```


1. 코드가 간결함

JSX를 사용하지 않은 코드

```
let element1 = React.createElement("div", null, "Hello");
```

JSX를 사용한 코드

```
let element2 = <div>Hello</div>
```

2) 가독성이 좋음

코드가 간단하여 개발자가 버그를 쉽게 발견할 수 있다.

3) Injection Attacks(XSS) 이라 불리는 해킹 방법을 방어

Injection Attacks(XSS)는 입력폼에서 문자나 숫자 값 대신 소스코드를 입력하여 그 코드가 실행되도록 만드는 해킹 방법이다. 리액트는 JSX를 렌더링할 때, JSX내에 포함된 모든 값을 문자열로 변환한다.

공격자가 입력폼에 악성코드를 넣어도, 그 코드가 실행되지 않고 단순한 텍스트로 처리된다.

새로운 파일을 생성할 때는 확장자를 .jsx로 선언한다.

js와 jsx는 모두 자바스크립트 파일로 기능을 동일하다.

순수 자바스크립트 파일은 .js 확장자를 사용하고, React의 JSX 문법을 사용한 파일은 .jsx를 사용한다.

이렇게 구분하면 다른 사람이 파일을 볼 때 어떤 기술이 사용되었는지 쉽게 파악할 수 있다.

 JS test.js test.jsx

JSX에서 자바스크립트를 사용하려면 중괄호 { } 로 표현할 수 있다.

이를 통해 HTML 태그의 속성이나 내용 안에서 자바스크립트 변수, 함수를 사용할 수 있다.

아래 코드에서 JSX 안에서 변수를 사용하면, 값으로 변환된다.

예제

```
const content = 'Hi';
const namePlaceholder = "이름을 입력하세요!";

return (
  <div>
    <div>{content}</div>
    <input placeholder={namePlaceholder}></input>
  </div>
);
```

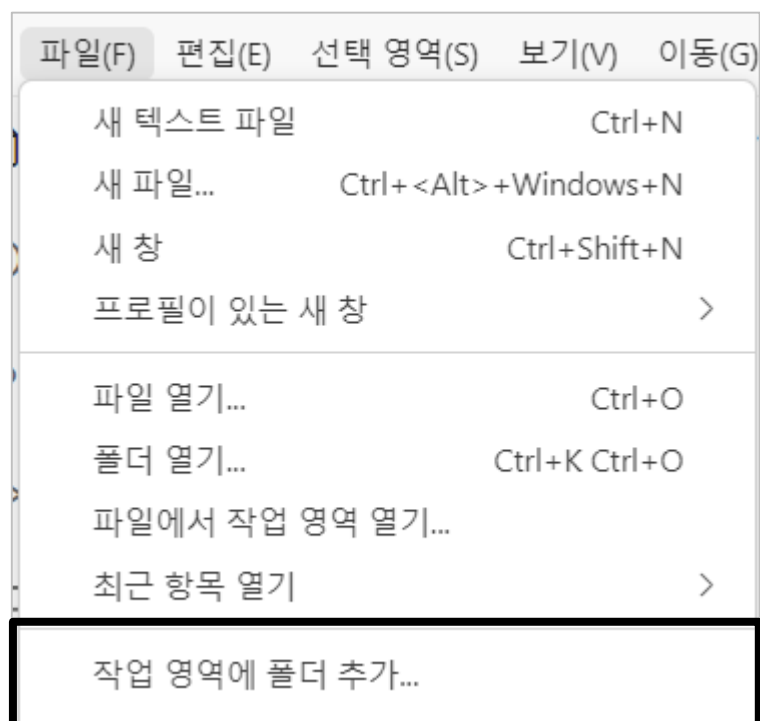
결과

```
<div>Hi</div>
<input placeholder="이름을 입력하세요!">
...
```

“react-app4” 이라는 이름으로 새로운 React 프로젝트를 생성한다.

```
npx create-react-app react-app4
```

생성한 프로젝트 폴더를 VSCode에 불러온다.



컴포넌트 만들기

컴포넌트란?

어플리케이션의 UI가 복잡해지면 UI 요소를 관리하는 것이 어려워진다.

버튼, 폼, 네비게이션 바 같은 UI 요소가 많아지면서 유지보수가 복잡해졌다.

컴포넌트는 이러한 문제를 해결하기 위해 UI를 독립적인 단위로 구분하는 방식이다.

컴포넌트란 복잡한 코드를 감추고 이름을 붙여 사용자 태그를 만드는 것이다.

React의 핵심 기능은 이러한 **사용자 정의 태그(컴포넌트)**를 생성하는 것이다.

```
<div className="App">
  <header>
    <h1><a href="/">Web</a></h1>
  </header>
  <ol>
    <li><a href="/read/1">html</a></li>
    <li><a href="/read/2">css</a></li>
    <li><a href="/read/3">js</a></li>
  </ol>
  <article>
    <h2>Welcome</h2>
    Hello, WEB
  </article>
</div>
```

=

```
<div className="App">
  <Header></Header>
  <Nav></Nav>
  <Article></Article>
</div>
```

컴포넌트 만들기

컴포넌트 만들기

App.js에 헤더, 내비게이션, 아티클 태그를 작성하여 간단한 페이지를 만든다.
이 세 영역을 각각 컴포넌트로 구성한다.

```
<div className="App">
  <header>
    <h1><a href="/">Web</a></h1>
  </header>
  <ol>
    <li><a href="/read/1">html</a></li>
    <li><a href="/read/2">css</a></li>
    <li><a href="/read/3">js</a></li>
  </ol>
  <article>
    <h2>Welcome</h2>
    Hello, WEB
  </article>
</div>
```

컴포넌트 만들기

컴포넌트 만들기

헤더 영역의 코드를 하나의 사용자 정의 태그로 만든다.

사용자 정의 태그는 대문자로 시작해야 하므로, Header 태그를 함수로 정의한다.

이전에 작성한 코드를 복사하여 Header 함수 안에 붙여 넣고, HTML 코드를 반환한다.

그리고 정의한 Header 태그를 사용하면 이전과 같은 결과가 출력된다.

```
function Header() {  
  return <header>  
    <h1><a href="/">Web</a></h1>  
  </header>  
}
```

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

1. Item 컴포넌트를 만들고, "Item Component" 라는 텍스트를 표시하세요.
2. App 컴포넌트 안에 Item 컴포넌트를 5번 추가하세요.

Item List

Item Component
Item Component
Item Component
Item Component
Item Component

1. Home, About, Contact 컴포넌트를 만들고, 각각 자신의 이름을 표시하세요.
2. Navbar 컴포넌트를 만들고, 그 안에 Home, About, Contact 컴포넌트를 추가하세요.
3. App 컴포넌트 안에 Navbar 컴포넌트를 추가하세요.

Navigation

Home

About

Contact

1. Content 컴포넌트를 만들고, "Content Component" 라는 텍스트를 표시하세요.
2. Section 컴포넌트를 만들고, 그 안에 Content 컴포넌트를 2번 추가하세요.
3. App 컴포넌트 안에 Section 컴포넌트를 2번 추가하세요.

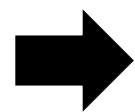
Section Component

Content Component
Content Component

Section Component

Content Component
Content Component

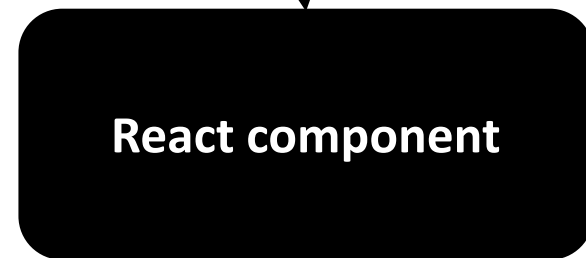
"react-app4"를 복사하여 "react-app5" 라는 이름으로 새로운 React 프로젝트를 생성한다.

 react-app4 react-app5

Props는 Properties 의 줄임말로 React 컴포넌트의 속성들을 의미한다.
부모 컴포넌트가 자식 컴포넌트에게 데이터를 전달할 때 Props를 사용한다.

일반 html 태그에는 src, width, height 속성들이 있으며, 이를 통해 태그의 동작과 모양을 정의할 수 있다.
예를 들어, src은 이미지 소스를 설정하고, width는 이미지의 크기를 설정한다.
이렇게 태그에 값을 입력하는 것을 속성이라고 한다.
React에서는 우리가 만든 컴포넌트에 props를 사용하여 속성을 부여할 수 있다.

Props



React component

React element

```
</img>
```

Props는 부모 컴포넌트가 자식 컴포넌트에 읽기 전용 값으로 전달한다.
따라서 자식 컴포넌트는 전달받은 props를 직접 변경할 수 없다.
만약 변경을 시도하면 TypeError가 발생한다.
Props를 변경하고 싶다면 부모 컴포넌트에서 다른 값을 전달해야 한다.

Uncaught runtime errors:

ERROR

```
Cannot assign to read only property 'title' of object '#<Object>'  
TypeError: Cannot assign to read only property 'title' of object '#<Object>'
```

Header 태그에 title 속성을 추가한다.

Header 함수에서 파라미터를 추가하고, 이름은 props라고 정의한다.

전달받은 파라미터를 사용하여 UI를 설정한다.

아래 예시에서 전달받은 props를 확인하면 "REACT"가 출력된다.

Header 태그

```
<Header title="REACT"></Header>
```

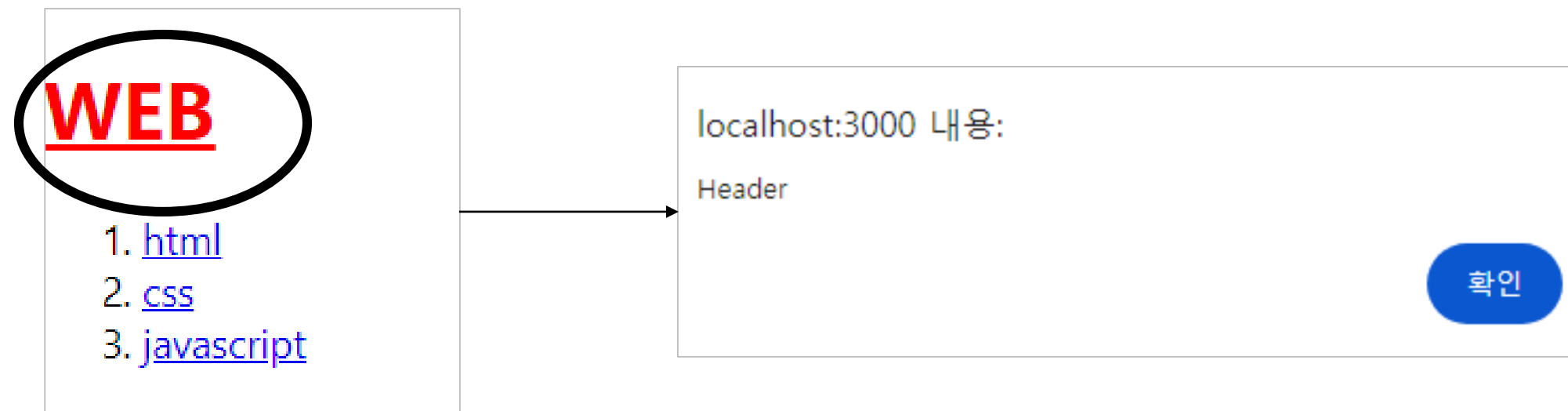
Header 함수

```
function Header(props) {  
  return <header>  
    <h1><a href="/">{props.title}</a></h1>  
  </header>  
}
```



```
▼ <header>  
  ▼ <h1>  
    <a href="/">REACT</a>  
  </h1>  
</header>
```

HTML을 보면 헤더에 a 태그가 있는데, 태그의 onclick을 사용하면 사용자가 클릭했을 때 경고창을 띄울 수 있다.
그런데 우리가 만든 컴포넌트는 이런 이벤트 기능이 없다.
그래서 우리가 만든 컴포넌트에서는 Props를 사용하여 이벤트를 설정할 수 있다.



헤더를 클릭했을 때 경고창이 뜨도록 이벤트를 처리하려면, `<Header>` 컴포넌트에 `onChangeMode` 함수를 props로 전달해야 한다. 그리고 Header 함수에서 props로 전달받은 `onChangeMode` 함수가 클릭 이벤트에서 실행되도록 설정한다.

Header 태그

```
<div>
  <Header onChangeMode={function () {
    alert('Header');
  }}></Header>
</div>
```

Header 함수

```
function Header(props) {
  return <header>
    <h1>
      <a href="/" onClick={event => {
        event.preventDefault();
        props.onChangeMode();
      }}>WEB</a>
    </h1>
  </header>
}
```


State란?

리엑트는 컴포넌트는 입력과 출력을 가지고 있다.

입력에는 prop이 있는데, prop은 컴포넌트에 값을 전달하고 전달된 값은 함수가 처리해서 UI로 반환된다.

state는 prop과 유사하게 컴포넌트를 다시 업데이트하는 요소이다.

하지만 state는 prop과 달리 컴포넌트 내부에서 관리된다.

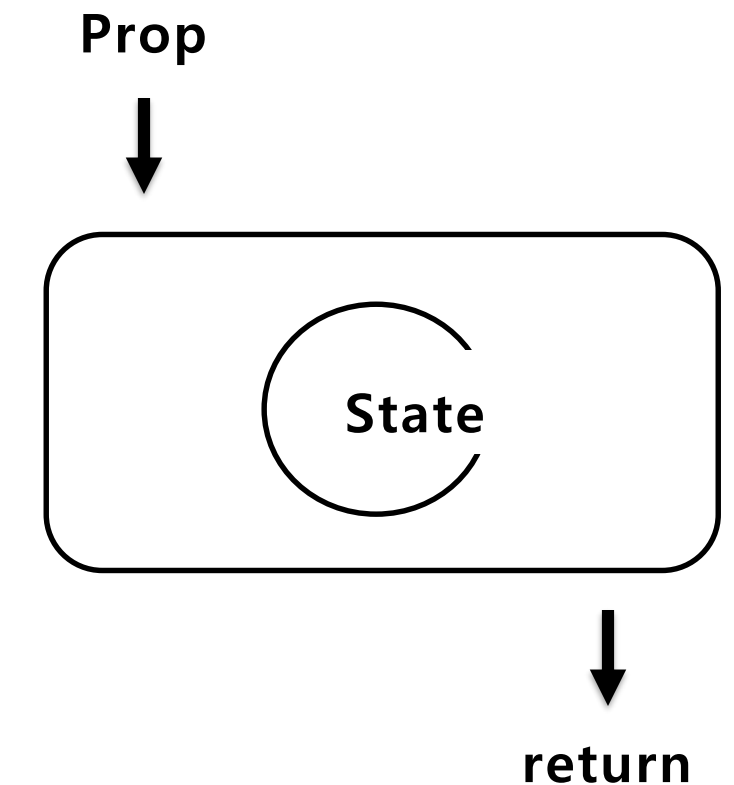
state가 변경되면 컴포넌트가 다시 렌더링되며 새로운 UI가 생성된다.

Prop와 State의 차이점

Prop는 외부에서 컴포넌트로 값을 전달할 때 사용한다.

State는 컴포넌트에서 보여줘야하는 내용이 사용자의 동작에 따라 바뀔 때 사용한다.

컴포넌트 내부에서 관리되며, 상태가 변경되면 컴포넌트가 다시 렌더링된다.



1. 렌더링과 관련된 값만 포함시켜야 한다.

State는 컴포넌트 UI에 영향을 미치는 값들만 포함해야 한다.

상태값이 변경되면 컴포넌트가 다시 렌더링되기 때문에, 불필요한 값을 state에 넣으면 불필요한 렌더링이 발생해서 성능에 영향을 미친다.

2. 함수를 사용하여 값을 변경해야 한다.

State는 직접 수정할 수 없고 반드시 React에서 제공하는 useState 함수를 사용해야 한다.

이를 통해 상태가 변경되면 React가 이를 감지하고 필요한 부분만 다시 렌더링할 수 있다.

헤더의 링크를 클릭하면 웰컴 페이지가 나타나고, 네비게이션을 클릭하면 해당 번호에 맞는 내용이 표시되도록 이벤트를 처리한다.

REACT

1. [html](#)
2. [css](#)
3. [javascript](#)

Welcome

Hello, web

REACT

1. [html](#)
2. [css](#)
3. [javascript](#)

html

html is ...

먼저 Mode라는 변수를 선언하고 mode의 값에 따라 다른 Article이 동적으로 생성되도록 설정한다.
클릭 이벤트가 발생하면 mode가 변경되도록 설정했지만, 이벤트가 발생해도 아무런 일이 발생하지 않는다.
Mode가 변경되더라도 App 함수가 다시 실행되지 않기 때문에 화면이 업데이트되지 않는 것이다.

```
let mode = 'WELCOME';
let content = null;

if(mode === 'WELCOME'){
  content = <Article title="Welcome" body="Hello, web"></Article>
}else if(mode === 'READ'){
  content = <Article title="Welcome" body="Hello, read"></Article>
}
```

```
return (
  <div className="App">
    <Header title="REACT" onChangeMode={()=>{
      mode = 'WELCOME';
    }}></Header>
    <Nav topics={topics} onChangeMode={()=>{
      mode = 'READ';
    }}></Nav>
    {content}
  </div>
);
```

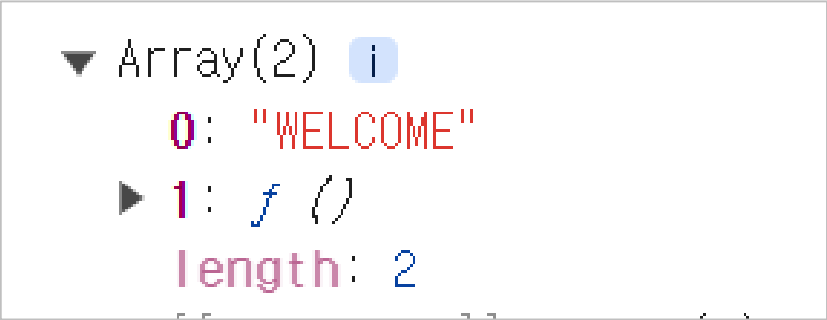
Mode가 변경될 때마다 컴포넌트가 다시 렌더링되도록 리액트에서 제공하는 useState 라는 hook을 사용해야 한다.

useState 혹은 상태값을 관리하고, 상태가 변경되면 React가 이를 감지하여 컴포넌트를 다시 렌더링한다. useState는 배열을 반환하며, 배열의 첫번째 요소는 현재상태, 두번째 요소는 상태를 변경하는 함수이다.

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

```
const mode = useState('WELCOME');
```

```
console.log(mode);
```



```
▼ Array(2) i  
  0: "WELCOME"  
  1: f ()  
  length: 2
```

setMode로 mode를 변경하면, mode가 변경되면서 App컴포넌트가 다시 실행된다.
그리고 변경된 content가 화면에 표시된다.

```
<Header title="REACT" onChangeMode={()=>{  
  setMode('WELCOME');  
}}></Header>  
<Nav topics={topics} onChangeMode={(id)=>{  
  setMode('READ');  
  setId(id);  
}}></Nav>
```

```
for(let t of topics){  
  console.log(t.id, id)  
  if(t.id === id ){  
    title = t.title;  
    body = t.body;  
  }  
}
```

```
content = <Article title={title} body={body}></Article>
```

html', '2. [css](#)', and '3. [javascript](#)'. At the bottom, the word 'html' is written in bold black text, and below it, the text 'html is ...' is visible. A red rectangle highlights the 'html' text and the text below it." data-bbox="592 476 768 871"/>

REACT

1. [html](#)
2. [css](#)
3. [javascript](#)

html

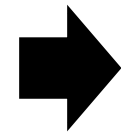
html is ...

지금까지 배운 내용을 활용하여 게시물 등록 기능을 구현한다.

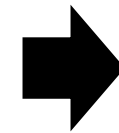
등록폼에서 사용자가 제목과 내용을 입력한 후, 등록 버튼을 클릭하면 네비게이션에 새로운 게시물이 추가된다.

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)



Create

[Create](#)

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)
4. [안녕](#)

등록 화면으로 이동하는 링크를 생성한다.

링크를 클릭하면 게시물 등록 페이지로 이동한다.

링크 클릭시 App 컴포넌트에 있는 mode 값을 'CREATE'로 변경하여 등록화면이 나타나도록 한다.

Mode 값이 변경되면 App 컴포넌트가 다시 렌더링되어 새로운 화면이 나타난다.

App 함수

```
<a href="/create" onClick={ event => {  
  event.preventDefault();  
  setMode('CREATE');  
}}>Create</a>
```

```
} else if(mode === "CREATE") {  
  content = <Create></Create>  
}
```

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)



Create

[Create](#)

게시물 등록화면을 Create 컴포넌트로 구현한다.

사용자 입력을 처리하기 위해 `<form>` 태그를 추가한다.

form 태그 안에 제목과 내용을 입력 받을 수 있는 필드를 추가한다.

Create 함수

```
function Create() {  
  return (  
    <article>  
      <h2>Create</h2>  
      <form>  
        <p><input type="text" name="title" />  
        <p><textarea name='body' placeholder='body' />  
        <p><input type="submit" value='Create' />  
      </form>  
    </article>  
  )  
}
```

Create

[Create](#)

게시물 등록 이벤트를 처리하기 위해 Create 컴포넌트에 함수를 전달한다.

Create 컴포넌트를 생성하는 부모 컴포넌트에서, 등록 버튼을 눌렀을 때 실행할 함수를 Prop으로 전달한다. 실행할 함수는 onCreate라는 이름으로 정의한다.

Create 컴포넌트에서는 폼을 제출하는 submit 버튼을 클릭했을 때 이벤트를 처리한다.

사용자가 입력한 제목과 내용을 꺼내고 Prop의 onCreate 함수를 실행한다.

onCreate 함수에 호출하여 사용자가 입력한 데이터를 전달한다.

이를 통해 부모 컴포넌트는 새로 작성된 게시물을 받아서 처리할 수 있다.

App 함수

```
} else if(mode === "CREATE") {  
  content = <Create onCreate={title, body=>{ }}></Create>  
}
```

Create 함수

```
function Create(props) {  
  return (  
    <article>  
      <h2>Create</h2>  
      <form onSubmit={event => {  
        event.preventDefault();  
        const title = event.target.title.value;  
        const body = event.target.body.value;  
        props.onCreate(title, body);  
      }}>  
    </form>  
  )  
}
```

새로운 글을 topics 배열에 추가하고 화면을 갱신한다.

topics 배열이 변경되면 화면이 다시 렌더링되도록 topics를 state로 만든다.

그리고 새로운 게시물을 topics 배열에 추가한다.

하지만 topics 배열에 새로운 원소를 추가해도 화면에는 변화가 없다.

topics 는 배열 타입이기 때문에, 배열에 원소를 추가해도 주소가 그대로 유지된다.

React는 상태(state)가 변경될 때만 화면을 다시 렌더링한다.

따라서 topics 배열의 복사본을 만들어서 새로운 게시물을 추가하고, 새로운 배열을 setState로 설정해야 한다.

App 함수

```
let [topics, setTopics] = useState([
  {id:1, title:'html', body:'html is ...'},
  {id:2, title:'css', body:'css is ...'},
  {id:3, title:'javascript', body:'javascript is ...'},
]);

else if(mode === "CREATE") {
  content = <Create onCreate={(_title, _body)=>{
    const newTopic = {id:nextId, title: _title, body: _body};
    const newTopics = [...topics];
    newTopics.push(newTopic);
    setTopics(newTopics);
  }} />
```

1. 등록화면에서 제목과 내용을 입력한 후, 등록 버튼을 클릭한다.
2. 네비게이션에 새로운 글이 추가되는지 확인한다.

Create

[Create](#)

WEB

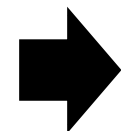
1. [html](#)
2. [css](#)
3. [javascript](#)
4. [안녕](#)

지금까지 배운 내용을 활용하여 게시물 등록 기능을 구현한다.

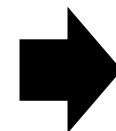
등록폼에서 사용자가 제목과 내용을 입력한 후, 등록 버튼을 클릭하면 네비게이션에 새로운 게시물이 추가된다.

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)



Create

[Create](#)

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)
4. [안녕](#)

Update

게시물 수정

이전에 등록한 게시물을 수정하는 기능을 구현한다.

사용자가 수정하려는 게시물을 선택하면 수정폼이 나타난다.

사용자가 제목이나 내용을 수정한 후, 저장 버튼을 클릭하면 기존 게시물이 업데이트된다.

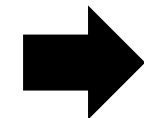
WEB

1. [html](#)
2. [css](#)
3. [javascript](#)

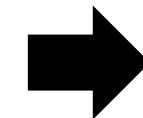
html

html is ...

- [Create](#)
- [Update](#)



Update



html

html is modify...

수정 화면으로 이동하는 링크를 생성한다.

링크를 클릭하면 게시물 수정페이지로 이동하게 된다.

먼저 변수로 선언하고, 변수에 수정 링크를 저장한다.

글을 수정할 때는 수정하려는 대상을 설정해야 하므로, 주소에 게시물의 ID값을 포함시킨다.

```
let contextControl = null;
```

```
} else if(mode === "READ") {  
  contextControl = <li><a href={'/update/' + id}>Update</a></li>;
```

수정 링크는 상세보기 화면에서만 나타나고 웰컴 화면에서는 보이지 않도록 설정한다.

조건문을 사용하여 mode의 값이 'READ'일 때만 링크가 나타나도록 한다.

조건문을 통해 읽기모드일 때만 수정링크가 화면에 표시된다.

```
<ul>
  <li>
    <a href="/create" onClick={ event => {
      event.preventDefault();
      setMode('CREATE');
    }}>Create</a>
  </li>
</ul>
{contextControl}
```

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)

Welcome

Hello, Web

- [Create](#)

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)

html

html is modify...

- [Create](#)
- [Update](#)

Update

수정 기능 구현하기

수정 링크를 클릭하면 수정화면이 나타난다.

링크를 클릭하면 Mode 값을 'UPDATE'로 변경한다.

content 변수에 Update 컴포넌트를 저장한다.

조건문을 사용하여 수정모드일때 content로 Update 컴포넌트를 출력하도록 설정한다.

```
} else if(mode === "UPDATE"){  
  content = <Update></Update>  
}
```

```
} else if(mode === "READ") {  
  contextControl = <li>  
    <a href={'/update/' + id} onClick={event=>{  
      event.preventDefault();  
      setMode('UPDATE');  
    }}>Update</a>  
  </li>;
```

Update

수정 기능 구현하기

게시물 수정화면을 Update 컴포넌트로 구현한다.

Create 함수에 있는 코드를 복사한 다음 붙여 넣는다.

함수의 매개변수에 props를 추가한다.

목적에 맞게 폼의 제목과 submit 버튼의 텍스트를 수정한다.

```
function Update(props) {  
  return (  
    <article>  
      <h2>Update</h2>  
      <form onSubmit={event => {  
        event.preventDefault();  
        const title = event.target.title.value;  
        const body = event.target.body.value;  
        props.onUpdate(title, body);  
      }}>  
      <p><input type="text" name="title" placeholder='title'></input>  
      <p><textarea name='body' placeholder='body'></textarea></p>  
      <p><input type="submit" value='Update'></input></p>  
    </article>  
  );  
}
```

Update

수정 기능 구현하기

게시물 수정을 위해 onCreate 부분을 onUpdate로 변경한다.

부모 컴포넌트에서, 수정 버튼을 눌렀을 때 실행할 함수를 Prop으로 전달한다.
실행할 함수는 onUpdate라는 이름으로 정의한다.

수정폼은 기존 게시물의 내용을 가지고 있어야 한다.

이를 위해 id값을 사용하여 제목과 내용을 찾아야 한다.

찾은 내용을 Update 컴포넌트에 prop으로 제목과 내용을 전달한다.

```
content = <Update onUpdate={(title, body)=>{}}></Update>
```

```
<form onSubmit={event => {  
  event.preventDefault();  
  const title = event.target.title.value;  
  const body = event.target.body.value;  
  props.onUpdate(title, body);  
}}>
```

Update

수정 기능 구현하기

Update 함수에서는 부모 컴포넌트로부터 전달받은 prop에서 제목과 내용을 꺼내고, 폼에 초기화한다.

App 함수

```
} else if(mode === "UPDATE"){  
  let title, body = null;  
  for(let t of topics){  
    console.log(t.id, id);  
    if(t.id === id ){  
      title = t.title;  
      body = t.body;  
    }  
  }  
  
  content = <Update title={title} body={body}  
  |}></Update>
```

Update 함수

```
<p><input type="text" name="title" value={props.title}  
<p><textarea name='body' value={props.body} placeholder  
<p><input type="submit" value='Update'></input></p>  
</form>
```

Update

Update

수정 기능 구현하기

수정폼에서 제목이나 내용을 바꿔보면 값이 변경되지 않는다.

왜냐면 폼에서 값을 입력해도, 부모에서 전달받은 prop는 값을 수정할 수 없기 때문이다.

값을 변경하기 위해 title과 body를 각각 state로 만들어 관리한다.

입력 필드에서 onChange 이벤트를 사용하여 상태를 업데이트한다.

입력 필드에서 사용자가 입력한 값을 꺼내어 set함수를 통해 state를 변경한다.

이제 사용자가 입력한 값이 상태로 저장되며, 폼에 바로 반영된다.

```
function Update(props) {  
  const [title, setTitle] = useState(props.title);  
  const [body, setBody] = useState(props.body);  
  
  <p>  
    <input type="text" name="title" value={title} onChange={event=>{  
      console.log(event.target.value);  
      setTitle(event.target.value);  
    }}></input>  
  </p>  
}
```

Update

Update

Update

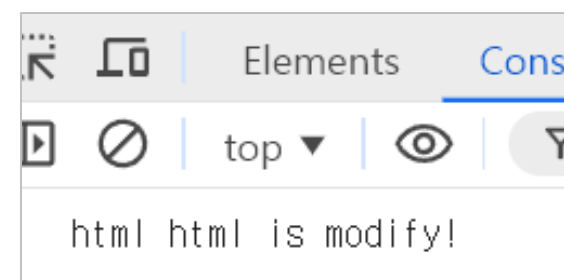
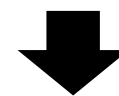
수정 기능 구현하기

수정 폼 안에 있는 Submit 버튼을 클릭하면, title과 body가 onUpdate 함수로 전달된다.
onUpdate 함수에서 전달받은 수정된 게시물 정보를 출력할 수 있다.

```
<form onSubmit={event => {  
  event.preventDefault();  
  const title = event.target.title.value;  
  const body = event.target.body.value;  
  props.onUpdate(title, body);  
}}>
```

```
content = <Update title={title} body={body} onUpdate={(title, body)=>{  
  console.log(title, body);  
}}>
```

Update



Update

수정 기능 구현하기

폼에서 입력받은 title과 body로 수정할 게시물 정보를 만든다.

선택한 게시물의 아이디와 함께 게시물의 정보를 객체로 생성한다.

topic 배열의 복제본을 만들고, 해당 요소를 찾아 수정된 게시물 정보로 교체한다.

글 수정이 끝났으면 상세보기 화면으로 이동한다.

```
content = <Update title={title} body={body} onUpdate={(title, body)=>{  
  const updateTopic = {id:id, title:title, body:body}  
  const newTopics = [...topics];  
  
  for(let i in newTopics){  
    if(newTopics[i].id === id){  
      newTopics[i] = updateTopic;  
      break;  
    }  
  }  
  setTopics(newTopics);  
  setMode('READ');  
}
```

1. 네비게이션에서 게시물을 선택한다.
2. 제목 또는 내용을 수정한다.
3. 수정 버튼을 클릭한 후, 게시물이 변경되었는지 확인한다.

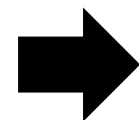
WEB

1. [html](#)
2. [css](#)
3. [javascript](#)

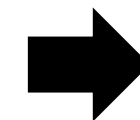
html

html is ...

- [Create](#)
- [Update](#)



Update



WEB

1. [html is](#)
2. [css](#)
3. [javascript](#)

html is

html is ...

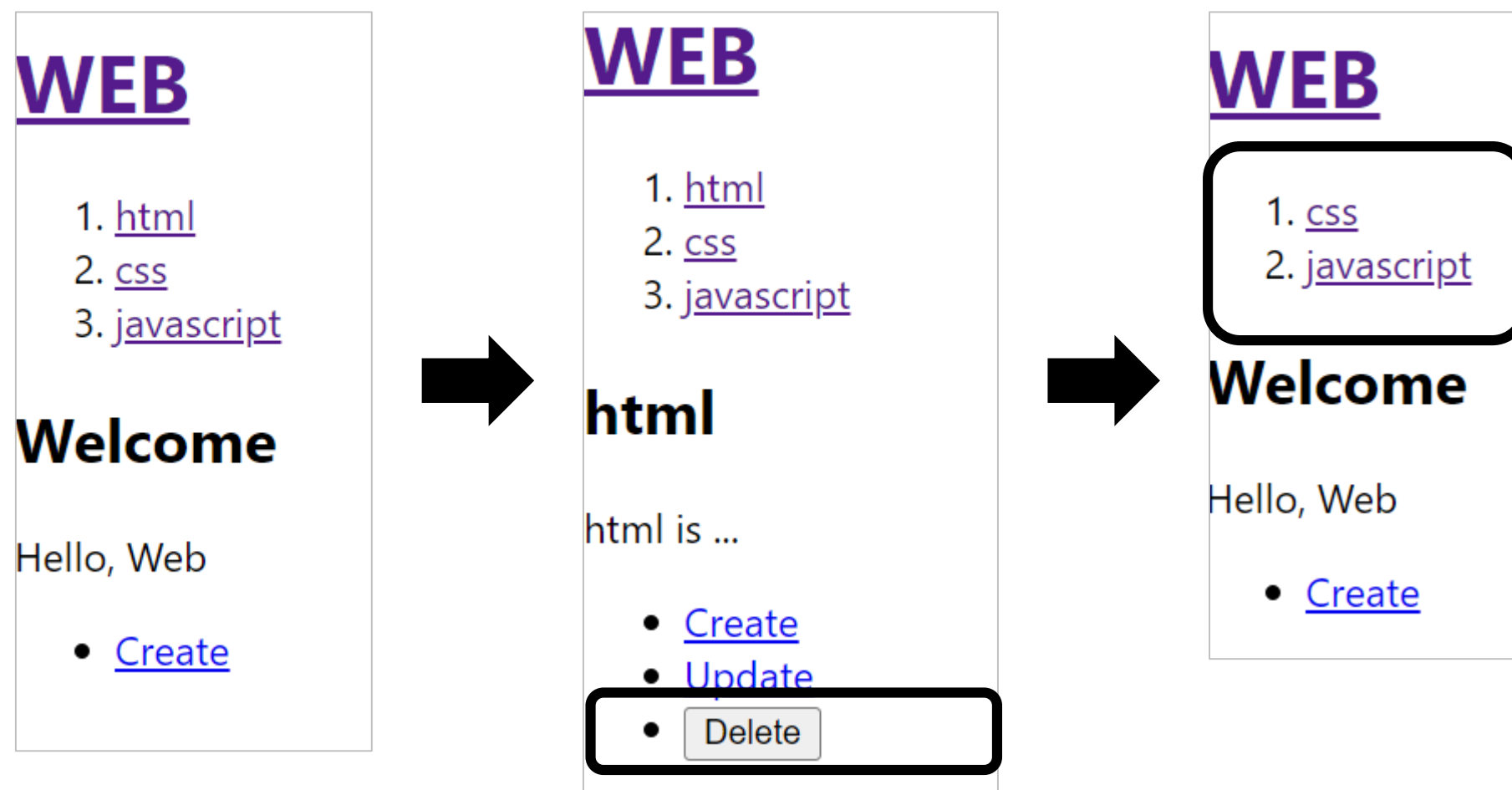
- [Create](#)
- [Update](#)

Delete

삭제 기능

게시물을 삭제하는 기능을 구현한다.

게시물 상세 화면에서 삭제 버튼을 클릭하면 기존 게시물이 삭제된다.



Delete

삭제 기능 구현하기

'READ' mode일 때, contextControl 변수에 Delete 버튼을 추가한다.
이제 게시물 상세 화면에 삭제버튼이 나타난다.

```
let contextControl = null;
```

```
} else if(mode === "READ") {  
  contextControl = <>  
    <li>  
      <a href={'/update/' + id} onClick={event=>{  
        event.preventDefault();  
        setMode('UPDATE');  
      }}>Update</a>  
    </li>  
    <li>  
      <input type='button' value='Delete' onClick={()=>{}}></input>  
    </li>  
  </>  
}
```

WEB

1. [html](#)
2. [css](#)
3. [javascript](#)

html

html is ...

- [Create](#)
- [Update](#)
-

Delete

삭제 기능 구현하기

Delete 버튼에 onClick 이벤트를 설정하여, topics 배열의 게시물 데이터를 삭제한다.
빈 배열을 생성하고 반복문을 사용하여 삭제할 게시물 제외하고 나머지를 요소를 저장한다.
SetTopics 함수를 사용하여 state를 변경한다.
글을 삭제 했기 때문에 상세 페이지로 이동할 수 없다.
웰컴 페이지로 이동하도록 mode를 'WELCOME'으로 설정한다.

```
<li>
  <input type='button' value='Delete' onClick={()=>{
    const newTopics = [];
    for(let i in topics){
      if(topics[i].id !== id){
        newTopics.push(topics[i]);
      }
    }
    setTopics(newTopics);
    setMode('WELCOME');
  }} />
</li>
```

Delete

테스트

1. 게시물을 선택하여 상세화면으로 이동한다.
2. 삭제버튼을 클릭한다.
3. 게시물이 삭제되고, 웰컴 페이지로 이동하는지 확인한다.

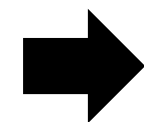
WEB

1. [html](#)
2. [css](#)
3. [javascript](#)

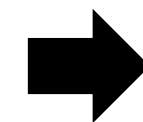
html

html is ...

- [Create](#)
- [Update](#)



Update



WEB

1. [html is](#)
2. [css](#)
3. [javascript](#)

html is

html is ...

- [Create](#)
- [Update](#)