

Hello,

KDT 웹 개발자 양성 프로젝트

1기! 38th

with







A-F 레벨 테스트
6년 차 아이돌의 재도전 플레디스

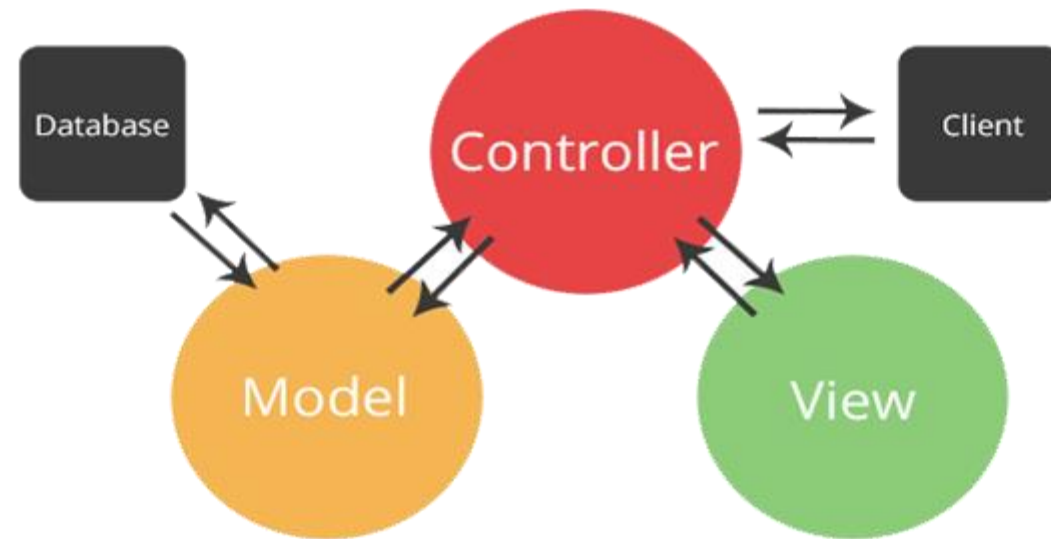


가희
댄스 트레이너
A 없습니다



React JS







홈 > 확장 프로그램 > Wappalyzer - Technology profiler



Wappalyzer - Technology profiler

✓ wappalyzer.com

★★★★★ 1,873 ⓘ | 개발자 도구 | 사용자 1,000,000+명

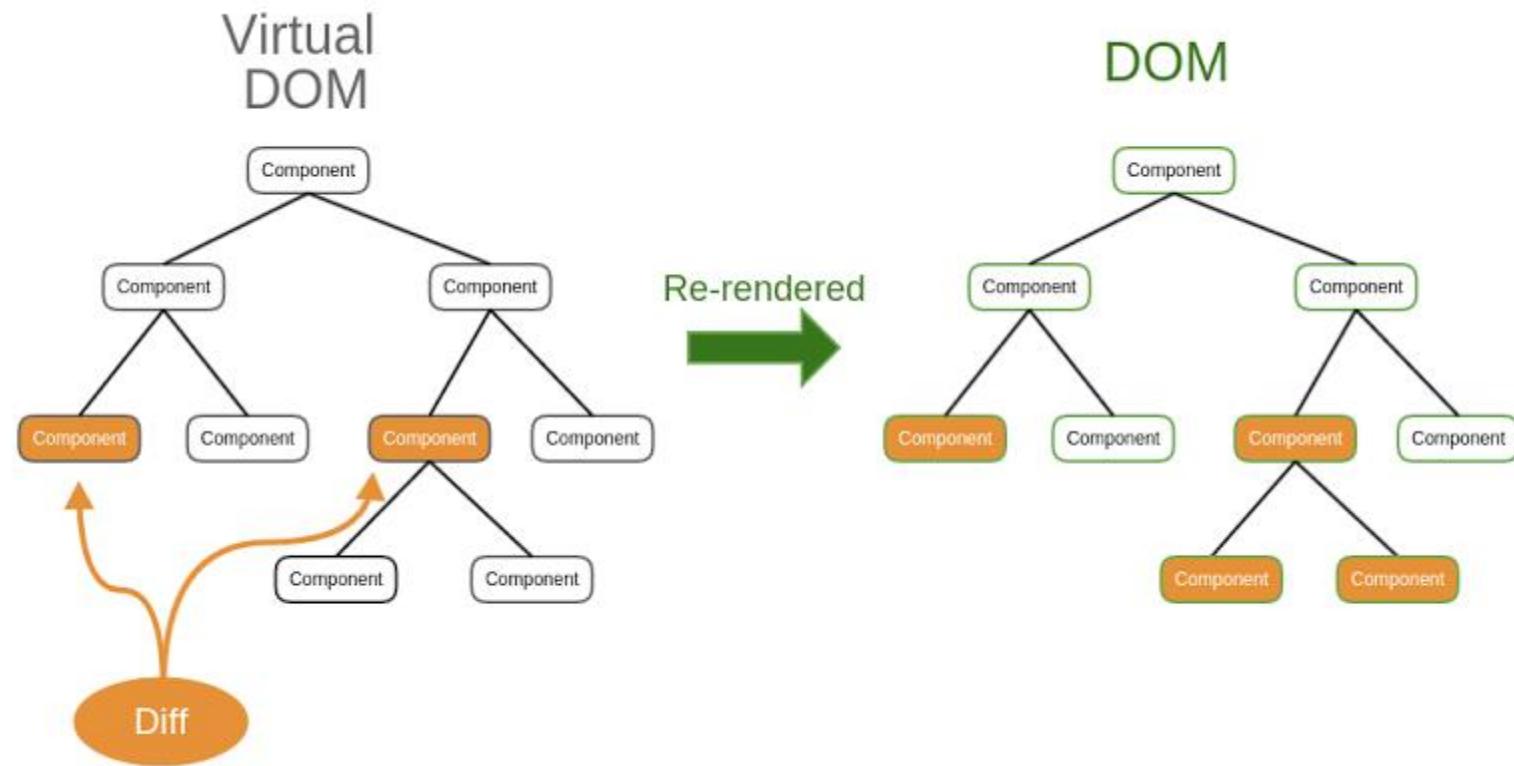
Chrome에서 삭제

<https://chrome.google.com/webstore/detail/wappalyzer-technology-pro/gppongmhjkpfnbhagpnmjfkannfbllamg>



Virtual DOM







JSX

(JavaScript XML)



```
function HelloWorldButton() {
  const [isClick, setClickState] = React.useState(false);
  const text = isClick ? "It's clicked" : "Hello, React world";

  return React.createElement(
    "button",
    { onClick: () => setClickState(!isClick) },
    React.createElement("div", null, React.createElement("span", null, text))
  );
}

const e = React.createElement;
const domContainer = document.querySelector("#app");
const root = ReactDOM.createRoot(domContainer);
root.render(e(HelloWorldButton));
```



```
// 함수형 컴포넌트
function HelloWorldButton() {
  const [isClick, setClickState] = React.useState(false);
  const text = isClick ? "It's clicked" : "Hello, React world";
  return (
    <button onClick={() => setClickState(!isClick)}>
      <div>
        <span>{text}</span>
      </div>
    </button>
  );
}

const domContainer = document.querySelector("#app");
const root = ReactDOM.createRoot(domContainer);
root.render(<HelloWorldButton />);
```

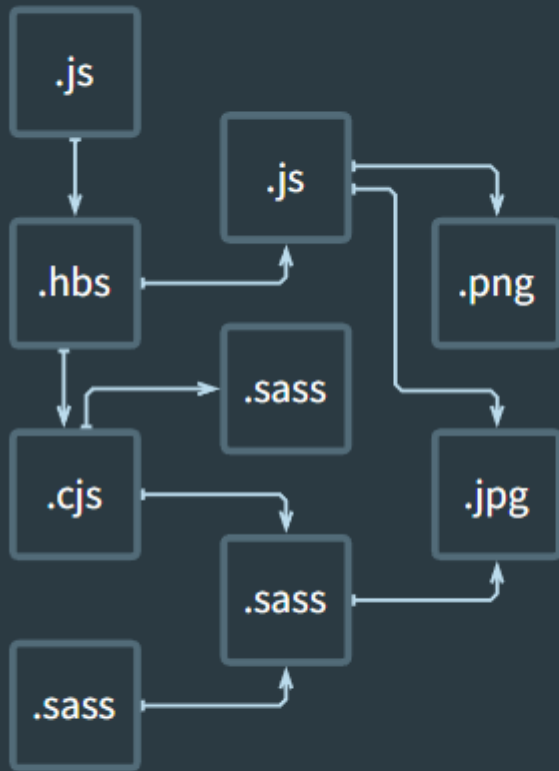


BABEL

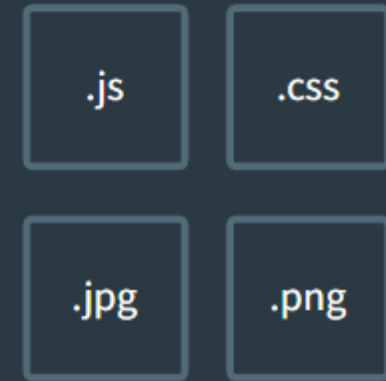
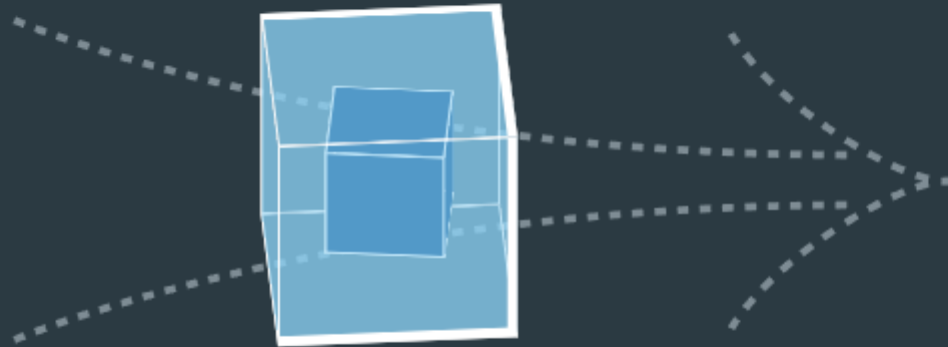


Webpack

bundle your images



MODULES WITH DEPENDENCIES



STATIC ASSETS



Create-react-app



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

[Learn React](#)



NPX?

NPM?



ws	101	websoc...	WebSocketClie...	0 B	Pendin
js.js	200	script	content.js:32	1.4 kB	3 n
dom.js	200	script	content.js:32	2.0 kB	2 n
js.js	200	script	content.js:32	1.4 kB	2 n
dom.js	200	script	content.js:32	2.0 kB	1 n
js.js	200	script	content.js:32	1.4 kB	2 n
12 requests 382 kB transferred 1.7 MB resources Finish: 6.28 s					



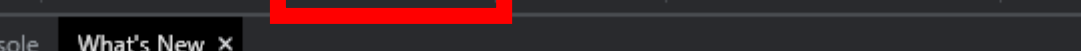
빌드 하기

- 프로젝트 폴더에 가서 빌드를 를 해봅시다!
- Npm run build

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/my-app (main)
$ npm run build

> my-app@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled successfully.
```



The screenshot shows the Chrome DevTools Performance tab. A red box highlights the text "166 kB resources" in the bottom status bar. Other status bar information includes "12 requests", "69.1 kB transferred", "Finish: 6.20 s", "DOMContentLoaded: 47 ms", and "Load: 58 ms". The bottom panel shows the "Console" tab with a "What's New" button.



Component



Component 의 종류



클래스형 컴포넌트

- 예전에 최초로 사용 되었던 컴포넌트 입니다!
- 컴포넌트 자체가 JS의 Class 와 유사하기 때문에 자연스럽게 사용 했었죠!
- 오래 된 만큼 state 와 라이프 사이클이라는 리액트의 장점을 사용 가능!
- 다만, 메모리 자원도 더 먹고 느리다는 단점이 있습니다
- 그리고 render 라는 함수를 사용해야만 그릴 수 있습니다!
- 최근에는 함수형 컴포넌트에게 완전히 밀리고 있습니다!



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

class ClassComponent extends Component {
  render() {
    return(
      <h1>Class Component 입니다.</h1>
    );
  }
}

export default ClassComponent;
```



함수형 컴포넌트

- JS에서 익숙하게 사용하였던 함수를 컴포넌트화 시킨 것 입니다!
- 아무래도 구조 자체가 클래스에 비해 단순하여 코드도 단순하고 빠르게 배울 수 있습니다
- 메모리도 덜 먹고 빠릅니다!(render 함수가 빠짐)
- 다만 예전에는 state 와 라이프사이클 기능 사용이 불가능하여 제한적으로 사용 → 최근에는 Hooks 라는 기능의 도입으로 같은 역할 수행 가능!
- 최근에는 대부분 이거만 씁니다!



```
const FunctionComponent = () => {  
  return <div>Funtional Component 입니다  
</div>  
};  
  
export default FunctionComponent;
```



State
(= Hook)



클래스형 컴포넌트의 State

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

```
class ClassState extends Component {
```

```
  // 현재 버전
```

```
  state = {  
    message: 0
```

```
};
```

```
render() {
```

```
  const { message } = this.state;
```

```
  const onClickEnter = () => { this.setState({ message: "안녕하세요!" }); };
```

```
  const onClickLeave = () => { this.setState({ message: "안녕히가세요!" }); };
```

```
  return (
```

```
    <div>
```

```
      <button onClick={onClickEnter}>입장</button>
```

```
      <button onClick={onClickLeave}>퇴장</button>
```

```
      <h1>{message}</h1>
```

```
    </div>
```

```
  );
```

```
}
```

```
}
```

```
export default ClassState;
```





함수형 컴포넌트의

State



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

function FuntionalState() {
  const [message, setMessage] = useState("");
  const onClickEnter = () => { setMessage("안녕하세요~"); };
  const onClickLeave = () => { setMessage("안녕히가세요."); };

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

export default FuntionalState;
```




오늘도 화이팅





State 활용하기



실습, 카운터 만들기!

- 최 상단에 <h1> 태그로 0 인 숫자가 있고, 아래에는 +1, -1 이라는 버튼이 각각 있습니다
- +1 버튼을 누르면 숫자가 +1 이 되고, -1 버튼을 누르면 숫자가 -1 이 되는 카운터 컴포넌트를 만들어 주세요!
- 함수형 컴포넌트로 만들어 주시면 됩니다!

0

+1 -1



Props



백엔드의 데이터를 프론트로!

- Ejs 에서 사용하던 render 함수를 기억 하시나요!?

```
res.render('board', {  
  ARTICLE,  
  articleCounts: articleLen,  
  userId: req.session.userId  
    ? req.session.userId  
    : req.user?.id  
    ? req.user?.id  
    : req.signedCookies.user,  
});
```



리액트에서는?

- 리액트에서는 프론트로 데이터를 어떤 방식으로 전달 할 수 있을까요?
- 리액트에서는 props(속성 properties 의 줄임말) 라는 것으로 손쉽게 전달이 가능합니다!
- JS 에서 매개변수를 전달 하듯이 props 로 데이터를 전달하면 바로 받아서 사용이 가능합니다!



MainHeader 에 props 적용

- MainHeader 의 매개변수 전달 부분에 {} 를 추가하고 사용할 props 명을 적어 주시면 됩니다!

```
import React from "react";

function MainHeader({ text }) {
  return (
    <h1>{text}</h1>
  )
}
```

```
export default MainHeader;
```

src/components/MainHeader.js



App.js 에서 props 사용하기

- App.js 의 컴포넌트 옆에 html 의 속성을 부여하는 것 처럼 props 명과 전달 하고자하는 데이터를 적어서 전달 하면 됩니다!

```
function App() {  
  return (  
    <div className="App">  
      <MainHeader text="Hello, props world!" />  
    </div>  
  );  
}
```

```
export default App;
```

src/App.js



```
function App() {  
  return (  
    <div className="App">  
      <MainHeader text="Hello, props world!" />  
      <MainHeader text="Bye Bye, props world!" />  
      <MainHeader text="Well come back, props world!" />  
    </div>  
  );  
}  
  
export default App;
```

src/App.js

Hello, props world!

Bye Bye, props world!

Well come back, props world!



다양한 데이터 받아오기!

- Props 는 다양한 데이터를 한꺼번에 받아 올 수 있습니다!
- 그리고 다양한 데이터는 props 라는 객체 하나로 받아서 사용이 가능합니다!

```
function App() {  
  return (  
    <div className="App">  
      <MainHeader text="Go naver" href="https://naver.com" userID="tetz!" />  
    </div>  
  );  
}
```

```
export default App;
```

src/App.js



```
import React from "react";

function MainHeader(props) {
  const { userID, text, href } = props;
  return (
    <div>
      <h1>{userID} 님 반갑습니다.</h1>
      <a href={href}>{text}</a>
    </div>

  )
}
```

src/components/MainHeader.js

```
export default MainHeader;
```



```
import React from "react";

function MainHeader({ userID, text, href }) {
  return (
    <div>
      <h1>{userID} 님 반갑습니다.</h1>
      <a href={href}>{text}</a>
    </div>

  )
}

export default MainHeader;
```

src/components/MainHeader.js



클래스형 컴포넌트

Props



클래스형 컴포넌트에서의 props

- 클래스형 컴포넌트인 만큼 기존의 props 로 접근하던 것을 this.props 로 접근 하시면 됩니다!
- 클래스이기 때문에 매개변수를 따로 받지 않습니다!
- This.props 를 쓰면 코드가 길어져서 보통 구조 분해 할당으로 변수로 만들어 사용합니다



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

class ClassProps extends Component {
  render() {
    const { name, age, nickName } = this.props;
    return (
      <div>
        <h1>이름: {name}</h1>
        <h2>나이: {age}</h2>
        <h2>별명: {nickName}</h2>
      </div>
    )
  }
}
```

```
export default ClassProps;
```

src/components/ClassProps.js



```
function App() {  
  return (  
    <div className="App">  
      <ClassProps name="뽀로로" age="5" nickName="사고뭉치" />  
    </div>  
  );  
}
```

src/App.js

이름: 뽀로로

나이: 5

별명: 사고뭉치



Props

활용하기



배열을 전달하고 props 로 받아서 처리!

- Props 로는 배열 같은 다양한 자료형의 전달이 가능합니다!
- 배열을 받아서 처리하는 CustomList.js 컴포넌트를 만들어 봅시다!

```
function CustomList(props) {  
  return (  
    <ul>  
      {props.arr.map((el) => {  
        return <li>{el}</li>  
      })}  
    </ul>  
  )  
}
```

```
export default CustomList;
```

src/components/CustomList.js



App.js 에서 배열을 전달

- App 에서 임의의 배열을 만들어서 전달하기!

```
function App() {  
  const nameArr = ['뽀로로', '루피', '크롱이'];  
  return (  
    <div className="App">  
      <CustomList arr={nameArr} />  
    </div>  
  );  
}
```

```
export default App;
```

src/App.js

뽀로로
루피
크롱이



객체를 전달하고 props 로 받아서 처리!

- 객체를 받아서 처리하는 CustomObj.js 컴포넌트를 만들어 봅시다!

```
function CustomObj(props) {  
  const { name, age, nickName } = props.obj;  
  return (  
    <div>  
      <h1>이름 : {name}</h1>  
      <h2>나이 : {age}</h2>  
      <h3>별명 : {nickName}</h3>  
    </div>  
  )  
}
```

```
export default CustomObj;
```

src/components/CustomObj.js



App.js 에서 객체를 전달

- App 에서 임의의 객체를 만들어서 전달하기!

```
function App() {  
  const pororoObj = {  
    name: "뽀로로",  
    age: "5",  
    nickName: "사고뭉치"  
  }  
  return (  
    <div className="App">  
      <CustomObj obj={pororoObj} />  
    </div>  
  );  
}
```

src/App.js

이름 : 뽀로로

나이 : 5

별명 : 사고뭉치



실습, props 와 state 활용하기!

- 프로필 변경하기 버튼을 클릭하면 props 로 전달 된, 오브젝트 배열의 값이 순서대로 변하는 ChangeObj.js 컴포넌트를 만들어 주세요!
- 전달되는 오브젝트의 배열은 다음 장의 App.js 코드를 참고해 주세요!
- App.js 에서는 boj
- 함수형으로 구현하기를 권해 드립니다!



컴포넌트 꾸미기



인라인으로 꾸미기!



인라인 스타일을 바로 적용

- JSX 문법을 통해 style 속성을 지정 가능하므로 스타일 속성 값을 객체로 선언한 다음 직접 삽입하는 방법도 가능합니다!
- 다만 아무래도 불편함이 많아서 실제로는 급한 상황이 아니면 사용하지 않습니다!



```
const divStyle = {  
  backgroundColor: "orange"  
}
```

src/style/TestCss.css

```
const headingStyle = {  
  color: "blue"  
}
```

```
const spanStyle = {  
  backgroundColor: "pink",  
  fontWeight: "700"  
}
```

```
export default function TestCss() {  
  return (  
    <div className="component-root" style={divStyle}>  
      <h1 style={headingStyle}>CSS 테스트 컴포넌트 입니다</h1>  
      <span style={spanStyle}>해당 컴포넌트를 기본 CSS로 꾸며 보아요!</span>  
    </div>  
  )  
}
```

CSS 테스트 컴포넌트 입니다

해당 컴포넌트를 기본 CSS로 꾸며 보아요!



기본 CSS로 꾸미기



기본 CSS 로 꾸미기!

- 컴포넌트의 결과물은 결국 html 코드이기 때문에 CSS 로 꾸미기가 가능합니다!
- 보통, src 폴더에 컴포넌트와 같은 이름의 css 파일을 만들어서 사용합니다!
- Src 폴더에 style 폴더를 만들고 거기에 컴포넌트와 같은 이름의 css 파일을 만들어 줍니다!
- 그리고 컴포넌트에 css 파일을 импорт 시켜주시면 됩니다!



```
div.component-root {  
  background-color: orange;  
}
```

```
div.component-root h1 {  
  color: red;  
}
```

```
div.component-root span {  
  background-color: white;  
  font-weight: 700;  
}
```

src/style/TestCss.css



```
import '../style/TestCss.css';

export default function TestCss() {
  return (
    <div className="component-root">
      <h1>CSS 테스트 컴포넌트 입니다</h1>
      <span>해당 컴포넌트를 기본 CSS로 꾸며 보아요!</span>
    </div>
  )
}
```

src/components/TestCss.js

CSS 테스트 컴포넌트 입니다

해당 컴포넌트를 기본 CSS로 꾸며 보아요!



Sass 사용하기!



SASS / SCSS 로 꾸미기!

- 리엑트는 기본적으로 컴포넌트에 컴포넌트가 들어가는 구조로 되어 있습니다! 그렇다 보니 중첩 문법을 지원하는 Sass 와의 궁합이 좋습니다!
- 그렇다보니 React 현업에서는 순수 css 보다 sass 또는 잠시 뒤에 배울 기술을 주로 활용합니다
- 그럼 Sass 를 사용해서 컴포넌트를 꾸며 보시죠!
- 먼저 Sass 설치 → `npm install node-sass`



SASS / SCSS 로 꾸미기!

- Src/style 폴더에 TestCss.scss 파일 생성

```
div.component-root {  
  background-color: skyblue;  
  & > h1 {  
    color: dodgerblue;  
  }  
  
  & > span {  
    background-color: pink;  
    font-weight: 700;  
  }  
}
```

src/style/TestCss.scss



```
import '../style/TestCss.scss';

export default function TestCss() {
  return (
    <div className="component-root">
      <h1>CSS 테스트 컴포넌트 입니다</h1>
      <span>해당 컴포넌트를 기본 CSS로 꾸며 보아요!</span>
    </div>
  )
}
```

src/components/TestCss.js

CSS 테스트 컴포넌트 입니다

해당 컴포넌트를 기본 CSS로 꾸며 보아요!



Styled Components



styled
components



Styled Components

- 리엑트는 기본적으로 컴포넌트 단위가 중심이 되는 구조입니다
- 컴포넌트는 상대적으로 작은 규모로 운영이 되기 때문에 기존의 방식처럼 css 파일을 분리해서 운영할 필요가 크지 않습니다!
- 따라서, 이전에 `<style>` 태그를 사용해서 css 를 썼던 것 처럼 사용하고 싶은 요구가 많았습니다!
- 그래서 탄생한 것이 Styled Components 입니다!



Styled Components 설치 및 불러오기

- 먼저 설치 부터 하시죠!
- Npm install styled-components
- 그리고 TestStyled.js 파일 만들기
- Styled 컴포넌트 모듈 불러오기

```
import styled from "styled-components";
```



Styled Components 사용하기

- Styled Components 는 독특하게 사용이 됩니다!

```
export default function TestStyled() {  
  return (  
    <MyDiv>  
      <MyHeading>h1 태그 입니다!</MyHeading>  
      <MySpan>span 태그 입니다!</MySpan>  
    </MyDiv>  
  )  
}
```

src/components/TestStyled.js

- 먼저 자기만의 이름으로 태그를 구성 합니다!



Styled Components 사용하기

- 각각의 태그를 변수에 할당하고 해당 태그의 실제적인 태그명을 styled 를 이용하여 지정합니다!

```
const MyDiv = styled.div`  
  background-color: orange;  
`;  
;
```

```
const MyHeading = styled.h1`  
  color: blue;  
`;
```

```
const MySpan = styled.span`  
  background-color: pink;  
  font-weight: 700;  
`;
```

src/components/TestStyled.js



Styled Components 사용하기

- 자신이 지정한 태그의 이름은 일종의 빈 태그로 만들어 지지만 styled 모듈을 사용하여 해당 태그의 실질적 태그를 지정할 수 있습니다
- 그리고 CSS 요소는 뒤에 따라오는 `` 백틱 문자의 내부에 써주시면 됩니다!



요소 보기



Styled Components 작동

- Styled Components 는 html 태그의 class 명을 기반으로 작동하며, 해당 class 명은 임의로 생성 됩니다

```
▼<div class="App">  
  ▼<div class="sc-bczRLJ jkucQt">  
    <h1 class="sc-gsnTZi b0qcB0">h1 태그 입니다!</h1>  
    <span class="sc-dkzDqf jmLKCS">span 태그 입니다!</span>  
  </div>
```



불편함 해결



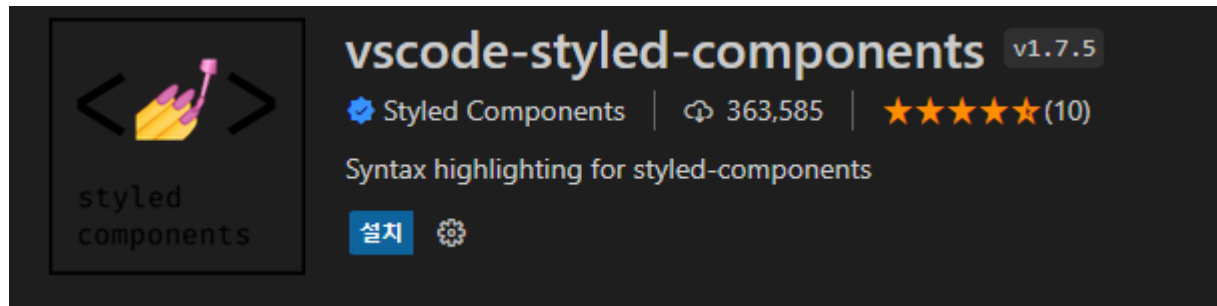
Styled Components 사용의 불편함

- Styled Components 사용시 CSS 를 입력하는 부분을 백틱 문자(` `) 내부에 작성을 해야합니다.
- 하지만 해당 부분은 JS에서 문자열로 취급 받기 때문에 CSS 의 스니펫을 활용할 수 없습니다!
- VS-Code 의 확장 프로그램을 설치하여 불편함을 해결 합니다!



Styled Components 사용의 불편함

- 확장 프로그램의 검색 창에 vscode-styled-components 입력

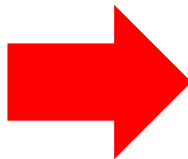


- 위의 확장 설치!

Styled Components 문법 표기



```
const MyDiv = styled.div`  
  background-color: orange;  
`;  
  
const MyHeading = styled.h1`  
  color: blue;  
`;  
  
const MySpan = styled.span`  
  background-color: pink;  
  font-weight: 700;  
`;
```



```
const MyDiv = styled.div`  
  background-color: orange;  
`;  
  
const MyHeading = styled.h1`  
  color: blue;  
`;  
  
const MySpan = styled.span`  
  background-color: pink;  
  font-weight: 700;  
`;
```


Styled Components snippet 기능!



```
const MyDiv = styled.div`
  background-color: orange;
  color: do;
  x
  dodgerblue
  darkolivegreen
const MyHeadi
  color: bl
  darkorange
  darkorchid
  darkgoldenrod
  darkviolet
const MySpan
  backgroun
  darksalmon
  darkturquoise
  font-weight: 700;
```





실습, React 초기 페이지를 Styled 로!

- 리액트 앱을 최초 생성 시, 만들어지는 app.js 컴포넌트를 App.css 가 아닌 Styled Components 를 이용해서 구현하기!
- 아래에 주어진 App.js 기본 코드와 Styled Components 를 위해 변경 된 코드를 비교하여 Styled Components 의 스타일을 적용시켜 주세요!
- 추가, 그림이 회전하는 keyframes 를 Styled Components 에서 적용하는 방법은 검색을 통해 구현해 주세요!



기본 App.js

```
import logo from './logo.svg';
import './App.css';

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
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```



Styled 적용을 위해 변경 된 App.js

```
import logo from './logo.svg';

function App() {
  return (
    <RootDiv>
      <AppHeader>
        <AppLogo src={logo} alt="app" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <MyA
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </MyA>
      </AppHeader>
    </RootDiv>
  );
}

export default App;
```

```
.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;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}
```

```
@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

App.css 코드



조건부 렌더링!



컴포넌트를 상황에 따라 켜고 끄기!

- 상황에 따라서 컴포넌트를 보여줄지 여부를 정해야 할 때가 있습니다!
- 그럴 때 사용하는 것이 조건부 렌더링 입니다!
- 이전 JS, HTML 에서는 Display 속성을 none 으로 해서 처리하곤 했습니다
- 리엑트는 JSX 문법을 사용 하므로, if 문 또는 3항 연산자, 논리 연산자와 HTML 태그를 같이 쓰면 되기 때문에 상당히 쉽습니다!
- 그럼 일단 한번 보시죠!



조건부 렌더링을 위한 컴포넌트 만들기

- Components 폴더에 ConditionalRender.js 파일 만들기

```
function ConditionalRender() {  
  return (  
    <h1>보이나요?</h1>  
  )  
}  
export default ConditionalRender;
```

src/components/ConditionalRender.js



App.js 에서 조건부 렌더링 처리

- App.js 에서 useState 를 활용해서 condition 에 따른 조건부 렌더링을 처리해 봅시다!
- Condition 이 감추기이면 ConditionalRender 를 보여주고, 보이기이면 감춰 봅시다!
- 이럴 땐, 보통 && 연산자를 사용합니다!



```
import ConditionalRender from './components/ConditionalRender';
import { useState } from 'react';

function App() {

  const [condition, setCondition] = useState("보이기");

  const onChange = () => {
    condition === "보이기" ? setCondition("감추기") : setCondition("보이기");
  }

  return (
    <div className="App">
      {condition === "감추기" && <ConditionalRender />}
      <button onClick={onChange}>{condition}</button>
    </div>
  );
}

export default App;
```

src/App.js



변수로 처리해서 코드 정리!

- 조건부 렌더링 자체를 변수에 넣어서 처리해 봅시다!
- 이렇게 하면 코드가 깔끔해 지는 효과가 있습니다!
- 이런 것이 가능한 이유는 바로! → JSX 문법 덕분이죠!



```
import ConditionalRender from './components/ConditionalRender';
import { useState } from 'react';

function App() {

  const [condition, setCondition] = useState("보이기");

  const onChange = () => {
    condition === "보이기" ? setCondition("감추기") : setCondition("보이기");
  }

  const conditionRender = condition === "감추기" && <ConditionalRender />;

  return (
    <div className="App">
      {conditionRender}
      <button onClick={onChange}>{condition}</button>
    </div>
  );
}

export default App;
```

src/App.js



실습, 조건부 렌더링 처리!

- PracticeOne, PracticeTwo 컴포넌트를 만들어 주세요!
- props 로 데이터를 받아서 h1 태그로 출력하는 간단한 구조를 가집시다!
- App.js 에서 버튼을 클릭하면 PracticeOne 과 PracticeTwo 가 번갈아서 렌더링 되는 조건부 렌더링 처리를 해주세요!
- 버튼의 컨텐츠도 렌더링 되는 컴포넌트가 나오게 해주시면 됩니다!



1번 컴포넌트

1번



2번 컴포넌트

2번



useRef



useRef? 이 친구는 또 뭐죠?

- 자 Input 태그를 사용해서 무언가를 입력 받는 상황을 그려 봅시다!
- 만약 이럴 때, useState 를 사용한다면?
- 인풋 태그 자체가 계속 리렌더링이 되어서 문제 + 낭비가 생길 겁니다!
- 이럴 때 쓰라고 있는 것이 useRef 되겠습니다!



useRef

- useRef 를 사용하면 참조하고자 하는 태그에 ref 속성을 주고 해당 태그의 변화를 감지할 수 있습니다 → 단, 리렌더링은 NoNo!
- 보통은 컴포넌트에 존재하는 인풋과 포커스를 관리하기 위해서 많이 사용됩니다!
- 즉, 인풋을 받을 때나 포커스가 이동 할 때에는 변화를 리액트 측에서 감지하고 있을 필요가 있지만 리렌더링은 필요가 없으니깐요!



JS 방식으로 구현하기

- Components 폴더에 TestRef.js 컴포넌트를 작성해 주세요!
- 인풋에 입력 된 값이, 상단에 있는 h1 태그의 컨텐츠가 되도록 구성하여 봅시다!
- 일단은, 기존의 JS 처럼 onChange 속성을 사용해서 이벤트 객체를 이용하여 구현하여 봅시다!



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

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

```
export default App;
```

src/App.js



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

export default function TestRef() {
  const [text, setText] = useState("안녕하세요!");

  const onChangeText = (e) => {
    const inputText = e.target.value;
    setText(inputText);
  }

  return (
    <div>
      <h1>{text}</h1>
      <input onChange={e => { onChangeText(e) }}></input>
    </div>
  )
}
```

dasdadadada

dasdadadada

src/components/TestRef.js



useRef 로 구현하기

- useRef 를 사용하면 참조하고자 하는 태그에 ref 속성을 주고 해당 태그의 변화를 감지할 수 있습니다 → 단, 리렌더링은 NoNo!
- 보통은 컴포넌트에 존재하는 인풋과 포커스를 관리하기 위해서 많이 사용됩니다!
- 즉, 인풋을 받을 때나 포커스가 이동 할 때에는 변화를 리액트 측에서 감지하고 있을 필요가 있지만 리렌더링은 필요가 없으니깐요!



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

export default function TestRef() {
  const [text, setText] = useState("안녕하세요!");

  const inputValue = useRef();

  const onChangeText = () => {
    setText(inputValue.current.value);
  }

  return (
    <div>
      <h1>{text}</h1>
      <input ref={inputValue} onChange={onChangeText}></input>
    </div>
  )
}
```

src/components/TestRef.js



useRef 값 확인하기

- useRef 로 설정한 값을 console.log 에 찍어 봅시다!

```
const onChangeText = () => {  
  console.log(inputValue);  
  setText(inputValue.current.value);  
}
```

```
▼ {current: input} ⓘ  
  ▼ current: input  
    value: "dsadad"  
    ▶ __reactEvents$3ew9d0gbjvx: Set(1) {'invalid__bubble'}  
    ▶ __reactFiber$3ew9d0gbjvx: FiberNode {tag: 5, key: null, el  
    ▶ __reactProps$3ew9d0gbjvx: {onChange: f}  
    ▶ _valueTracker: {getValue: f, setValue: f, stopTracking: f}  
    ▶ _wrapperState: {initialChecked: undefined, initialValue: '...
```




useRef

focus



useRef 로 포커스 이동!

- useRef 가 자주 사용되는 포커스 사용 방법을 알아 봅시다!
- 컴포넌트 폴더에 ChangeFocus.js 컴포넌트를 만들어 봅시다!
- 2개의 인풋 태그를 만들고 해당 인풋에 ref 속성을 부여!
- useRef 로 각각 인풋의 속성 값을 변수에 담고 해당 변수를 통해 input 태그에 포커스를 부여해 봅시다!
- 해당 값에 대한 접근은 current 객체를 통해 해야합니다!



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

export default function TestRef() {
  const input1 = useRef();
  const input2 = useRef();

  const changeFocusOne = () => {
    input1.current.focus();
  }

  const changeFocusTwo = () => {
    input2.current.focus();
  }

  return (
    <div>
      <input ref={input1}></input>
      <input ref={input2}></input>
      <br></br>
      <button onClick={changeFocusOne}>1</button>
      <button onClick={changeFocusTwo}>2</button>
    </div>
  )
}
```

src/components/ChangeFocus.js



수고하셨습니다!