리액트 컴포넌트 스타일링하기

리액트에서 컴포넌트를 스타일링 할 때 다양한 기술이 사용될 수 있다. 리액트에서 컴포넌트를 스타일링 하는 가장 기본적인 방법은 css 파일을 만들어서 컴포넌트에서 import해서 사용하는 것이다. 다음은 리액트 프로젝트에서 컴포넌트를 스타일링 할 때 자주 사용되는 기술들이다.

1. Sass
2. CSS Module
3. styled-components

Sass(Syntactically Awesome Style Sheets)

Css pre-processor로서, 복잡한 작업을 쉽게 할 수 있게 해주고, 코드의 재활용성을 높여줄 뿐만 아니라 코드의 가독성을 높여주어 유지보수를 쉽게 해준다.

CSS Module

리액트 프로젝트에서 컴포넌트를 스타일링 할 때 CSS Module이라는 기술을 사용하면, Css 클래스가 중첩되는 것을 완벽히 방지할 수 있다. 만드는 방법은 css 파일의 확장자를 .module.css 로 하면 된다.

CSS Module은 클래스 이름에 대하여 고유한 이름들이 만들어지기 때문에, 실수로 CSS 클래스 이름이 다른 관계없는 곳에서 사용한 CSS 클래스 이름과 중복되는 일에 대하여 걱정할 필요가 없다. 다음과 같은 상황에 사용하면 유용하다.

* 레거시 프로젝트에 리액트를 도입할 때 (기존 프로젝트에 있던 CSS 클래스와 이름이 중복되어도 스타일이 꼬이지 않게 해줍니다.)
* CSS 클래스를 중복되지 않게 작성하기 위하여 CSS 클래스 네이밍 규칙을 만들기 귀찮을 때

새로운 프로젝트를 생성하자

$ npx create-react-app styling-with-css-module

프로젝트 에디터를 열고 src 디렉터리에 components 디렉터리를 만든 후, 그 안에 CheckBox.js를 생성하자.

components/CheckBox.js

import React from 'react';

function CheckBox({ children, checked, ...reset }) {

    return (

        <div>

            <label>

                <input type="checkbox" checked={checked} {...reset}/>

                <div>{checked ? '체크됨' : '체크 안됨'}</div>

            </label>

            <span>{children}</span>

        </div>

    );

}

export default CheckBox;

여기서 …reset을 사용한 이유는, CheckBox 컴포넌트에게 전달하게 될 name, onChange 같은 값을 그대로 input에 넣어주기 위함이다.

App.js

import React,{useState} from 'react';

import CheckBox from './components/CheckBox';

function App() {

  const [check.setCheck]=useState(false);

  const onChange=e=>{

    setCheck(e.target.checked);

  };

  return (

    <div>

     <CheckBox onChange={onChange} checked={check}>

       다음 약관에 모두 동의

     </CheckBox>

     <p>

       <b>check:</b>

       {check ? 'true':'false'}

     </p>

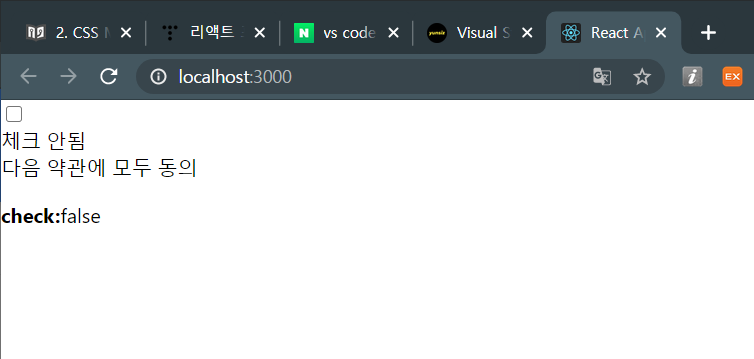
    </div>

  );

}

export default App;

를 다음과 같이 수정하고 yarn start명령어를 통해 서버를 열자.



다음과 같은 결과를 얻을 수 있다. 지금은 input 부분을 선택했는데도 값이 바뀌는데 스타일링을 통해 체크박스를 활용하자.

react-icons라는 라이브러리를 설치하면 아이콘들을 컴포넌트 형태로 쉽게 사용할수 있다.

$yarn add react-icons 를 입력하자

그리고 CheckBox.module.css 파일을 components 디렉터리에 생성하여 스타일링해보자

components/CheckBox.module.css

.checkbox{

    display: flex;

    align-items:center;

}

.checkbox label{

    cursor: pointer;

}

.checkbox input{

    width:0;

    height:0;

    position:absolute;

    opacity:0;

}

.checkbox span{

    font-size:1.125rem;

    font-weight:bold;

}

.icon{

    display:flex;

    align-items:center;

    font-size:2rem;

    margin-right:0.25rem;

    color:#adb5bd;

}

.checked{

    color: #339af0;

}

checkbox 컴포넌트를 다음과 같이 수정하자

components/CheckBox.js

import React from 'react';

import {MdCheckBox,MdCheckBoxOutlineBlank} from 'react-icons/md';

import styles from './CheckBox.module.css';

function CheckBox({ children, checked, ...reset }) {

    return (

        <div className={styles.checkbox}>

            <label>

                <input type="checkbox" checked={checked} {...reset}/>

                <div className={styles.icon}>

                    {checked ? (

                    <MdCheckBox className={styles.checked}/>)

                     : (<MdCheckBoxOutlineBlank/>)}</div>

            </label>

            <span>{children}</span>

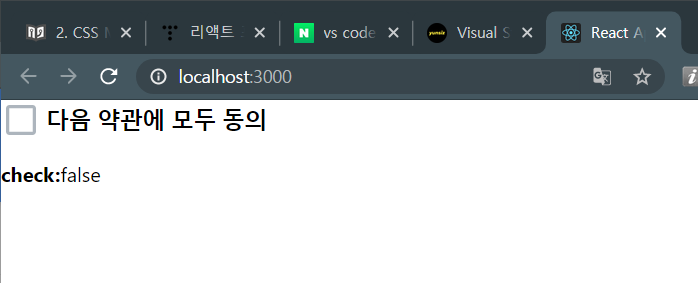
        </div>

    );

}

export default CheckBox;

다음과 같이 수정하면



위와 같은 결과를 얻을 수 있고, text를 선택해도 박스가 체크되지 않는다.

이때 개발자 도구로 엘리먼트를 선택해보면 고유한 클래스 이름이 만들어진 것을 확인할 수 있다.

CSS Module을 사용할때에는 styles.icon 이런식으로 객체안에 있는 값을 조회해야 하는데, 만약 클래스 이름에 – 가 들어가 있다면 다음과 같이 styles [‘my-class’] 라고 작성해야한다.

그리고 만약 여러 개가 있다면 다음과 같이 작성해야한다.

${styles.one} ${styles.two}

조건부 스타일링을 해야한다면 ${styles.one} ${condition?styles.two : ‘’}

classnames 라이브러리에 bind 기능이 있는데, 이 기능을 사용하면 css module을 조금 더 편하게 사용할 수 있다

$yarn add classnames 이후에

checkbox.js를 다음과 같이 수정하자

import React from 'react';

import { MdCheckBox, MdCheckBoxOutlineBlank } from 'react-icons/md';

import styles from './CheckBox.module.css';

import classNames from 'classnames/bind';

const cx = classNames.bind(styles);

function CheckBox({ children, checked, ...rest }) {

return (

<div className={cx('checkbox')}>

<label>

<input type="checkbox" checked={checked} {...rest} />

<div className={cx('icon')}>

{checked ? (

<MdCheckBox className={cx('checked')} />

) : (

<MdCheckBoxOutlineBlank />

)}

</div>

</label>

<span>{children}</span>

</div>

);

}

export default CheckBox;

styled-components

이는 CSS in JS라는 기술이다. JS안에 CSS를 작성하는 것인데, styled-componets를 사용하기 전에 Tagged Template Literal 이라는 문법에 대해 짚고 넘어가자. 완벽히 이해하지 않아도 앞으로 styled-components를 사용하는데 전혀 지장이 가지 않으니 가볍게 이해하자.

Template Literal은 문자열 조합을 더욱 쉽게 할 수 있게 해주는 ES6 문법이다.

이제 본격적으로 **styled-components를 사용**해보자

새로운 프로젝트를 만들고, styled-components를 설치하자

$ npx create-react-app styling-with-styled-components

$ cd styling-with-styled-components

$ yarn add styled-components

다음과 같이 App.js를 수정하자

import React from 'react';

import styled from 'styled-components';

const Circle = styled.div`

  width: 5rem;

  height: 5rem;

  background: black;

  border-radius: 50%;

`;

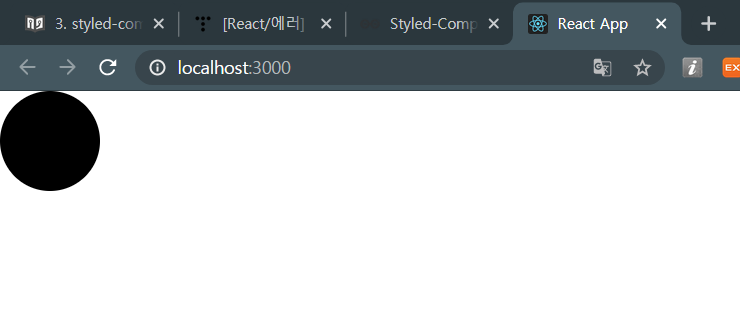
function App() {

  return <Circle />;

}

export default App;

아래와 같은 결과를 얻을 수 있다.



styled-components를 사용하면 이렇게 스타일을 입력함과 동시에 해당 스타일을 가진 컴포넌트를 만들 수 있다. 만약에 div를 스타일링 하고 싶다면 styled.div, input을 스타일링 하고 싶으면 styled.input 이런식으로 사용하면 된다.

이번에는, Circle 컴포넌트에 color라는 props를 넣어주자.

import React from 'react';

import styled from 'styled-components';

const Circle = styled.div`

  width: 5rem;

  height: 5rem;

  background: ${props=>props.color||'black'};

  border-radius: 50%;

`;

function App() {

  return <Circle color="blue"/>;

}

export default App;

위와 같이 작성하면 원이 파란색으로 바뀌는데, Circle 컴포넌트에서는 color props 값을 설정해주었으면 해당 값을 그렇지 않으면 검정색을 배경색으로 사용하도록 설정한 것이다.

이번에는 huge라는 props를 설정했을 때 크기를 더 키워서 보여주도록 하는 작업을 해보자

import React from 'react';

import styled, {css} from 'styled-components';

const Circle = styled.div`

  width: 5rem;

  height: 5rem;

  background: ${props=>props.color||'black'};

  border-radius: 50%;

  ${props=>props.huge && css`

width:10rem;

height:10rem;

`}

`;

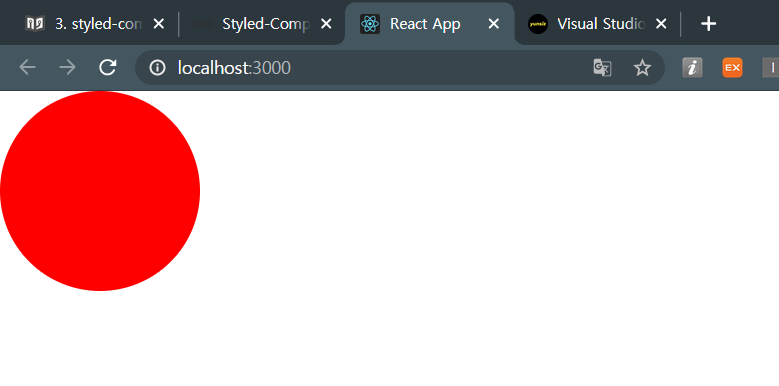
function App() {

  return <Circle color="red" huge/>;

}

export default App;

아래와 같이 거대한 빨간색 원이 나타난다.



이런식으로 여러 줄의 CSS코드를 조건부로 보여주고 싶다면 *css*를 사용해야 한다. *css*를 불러와서 사용을 해야 그 스타일 내부에서도 다른 proops를 조회할 수 있다.

**Button 만들기**

src안에 components 디렉터리를 생성 후, 그 안에 button.js파일을 생성하자.

import React from 'react';

import styled from 'styled-components';

const StyledButton=styled.button`

display:inline-flex;

outline:none;

border:none;

border-radius:4px;

color:white;

font-weight:bold;

cursor:pointer;

padding-left:1rem;

padding-right:1rem;

height:2.25rem;

font-size:1rem;

background:##228be6;

&:hover{

    background:#339af0;

}

&:active{

    background:#1c7ed6;

}

&+&{

    margin-left:1rem;

}

`;

function Button({children,...rest}){

    return <StyledButton {...rest}>{children}</StyledButton>;

}

export default Button;

그 다음엔 App.js에서 Button을 사용해보자

import React from 'react';

import styled from 'styled-components';

const StyledButton=styled.button`

display:inline-flex;

outline:none;

border:none;

border-radius:4px;

color:white;

font-weight:bold;

cursor:pointer;

padding-left:1rem;

padding-right:1rem;

height:2.25rem;

font-size:1rem;

background:##228be6;

&:hover{

    background:#339af0;

}

&:active{

    background:#1c7ed6;

}

&+&{

    margin-left:1rem;

}

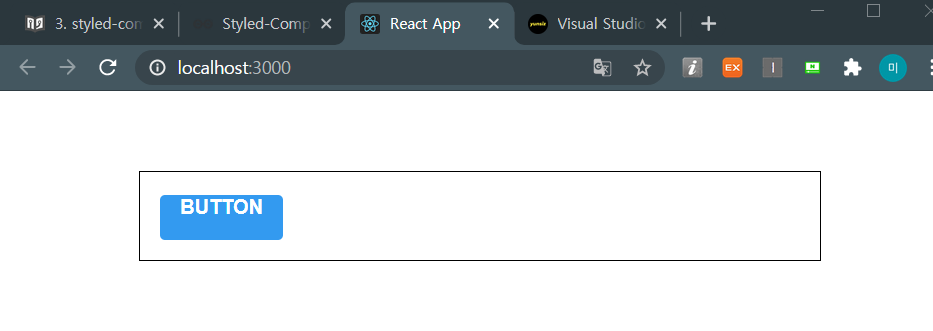
`;

function Button({children,...rest}){

    return <StyledButton {...rest}>{children}</StyledButton>;

}

export default Button;



**polished의 스타일 관련 유틸 함수 사용하기**

CSS in JS에서도 유틸 함수를 사용하고 싶다면 *polished*라는 라이브러리를 사용하면 된다.

$yarn add polished

를 통해 패키지를 설치한 후 기존에 색상 부분을 polished의 유틸 함수들로 대체해보자

button.js

import React from 'react';

import styled from 'styled-components';

import {darken,lighten} from 'polished';

const StyledButton=styled.button`

display:inline-flex;

outline:none;

border:none;

border-radius:4px;

color:white;

font-weight:bold;

cursor:pointer;

padding-left:1rem;

padding-right:1rem;

height:2.25rem;

font-size:1rem;

background:##228be6;

&:hover{

    background:${lighten(0.1,'#228be6')};

}

&:active{

    background:${darken(0.1,'#228be6')};

}

&+&{

    margin-left:1rem;

}

`;

function Button({children,...rest}){

    return <StyledButton {...rest}>{children}</StyledButton>;

}

export default Button;

커서를 버튼에 올렸을 때 눌렀을 때 색상이 바뀜을 알 수 있다.

이제 회색, 핑크색 버튼들을 만들어보자. 색상 코드를 지닌 변수를 button.js에서 선언하는 대신 *ThemeProvider*라는 기능을 사용하여 styled-components로 만드는 모든 컴포넌트에서 조회하여 사용할 수 있는 전역적인 값을 설정해보자

App.js를 다음과 같이 수정해보자.

import React from 'react';

import Button from './components/button';

import styled,{ThemeProvider} from 'styled-components';

const AppBlock = styled.div`

  width: 512px;

  margin:0 auto;

  margin-top:4rem;

  border:1px solid black;

  padding:1rem;

`;

function App() {

  return(

  <ThemeProvider

  theme={{

    palette:{

      blue:'#228be6',

      gray:'#495057',

      pink:'f06595'

    }

  }}

  >

    <AppBlock>

      <Button>BUTTON</Button>

    </AppBlock>

  </ThemeProvider>

  );

}

export default App;

이렇게 해서 theme을 설정하면 ThemeProvider 내부에 렌더링된 styled-components로 만든 컴포넌트에서 palette를 조회하여 사용할 수 있다. 한 번 Button 컴포넌트에서 선언한 palette.blue값을 조회해보자.

Button.js

import React from 'react';

import styled,{css} from 'styled-components';

import {darken,lighten} from 'polished';

const StyledButton=styled.button`

display:inline-flex;

outline:none;

border:none;

border-radius:4px;

color:white;

font-weight:bold;

cursor:pointer;

padding-left:1rem;

padding-right:1rem;

height:2.25rem;

font-size:1rem;

${props=>{

    const selected=props.theme.palette.blue;

    return css`

        background:${selected};

        &:hover{

            background:${lighten(0.1,selected)};

        }

        &:active{

            background:${darken(0.1,selected)};

        }

        `;

}}

&+&{

    margin-left:1rem;

}

`;

function Button({children,...rest}){

    return <StyledButton {...rest}>{children}</StyledButton>;

}

export default Button;

ThemeProvider로 설정한 값은 styled-components에서 props.theme로 조회할 수 있다. 지금은 selected 값을 무조건 blue 값을 가르키게 했는데, 이 부분을 button 컴포넌트가 color props를 통하여 받아오게 될 색상을 사용하도록 수정해보자.

button.js

import React from 'react';

import styled,{css} from 'styled-components';

import {darken,lighten} from 'polished';

const StyledButton=styled.button`

display:inline-flex;

outline:none;

border:none;

border-radius:4px;

color:white;

font-weight:bold;

cursor:pointer;

padding-left:1rem;

padding-right:1rem;

height:2.25rem;

font-size:1rem;

${props=>{

    const selected=props.theme.palette[props.color];

    return css`

        background:${selected};

        &:hover{

            background:${lighten(0.1,selected)};

        }

        &:active{

            background:${darken(0.1,selected)};

        }

        `;

}}

&+&{

    margin-left:1rem;

}

`;

function Button({children,...rest}){

    return <StyledButton {...rest}>{children}</StyledButton>;

}

Button.defaultProps={

    color:'blue'

};

export default Button;

지금은 기본 색상이 blue가 되도록 설정해주었다. 이제 App 컴포넌트를 열어서 회색,핑크색 버튼을 렌더링하자.

App.js

import React from 'react';

import Button from './components/button';

import styled,{ThemeProvider} from 'styled-components';

const AppBlock = styled.div`

  width: 512px;

  margin:0 auto;

  margin-top:4rem;

  border:1px solid black;

  padding:1rem;

`;

function App() {

  return(

  <ThemeProvider

  theme={{

    palette:{

      blue:'#228be6',

      gray:'#495057',

      pink:'#f06595'

    }

  }}

  >

    <AppBlock>

      <Button>BUTTON</Button>

      <Button color="gray">BUTTON</Button>

      <Button color="pink">BUTTON</Button>

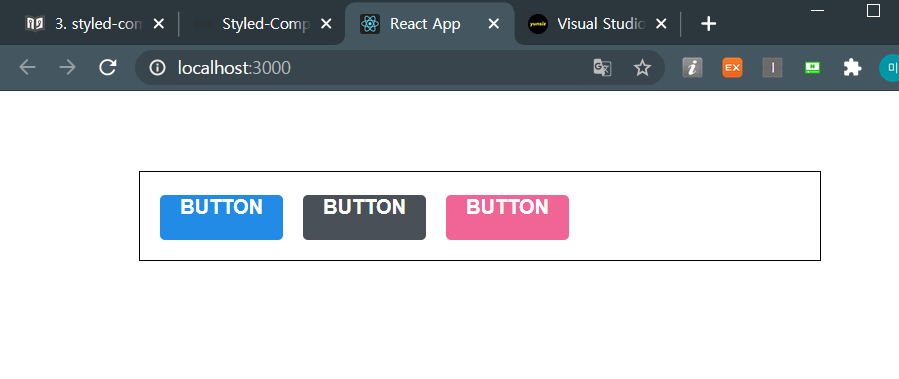
    </AppBlock>

  </ThemeProvider>

  );

}

export default App;



button.js는 props.theme.palette.blue 이런식으로 값을 조회하는 대신에 비구조화 할당 문법을 사용하여 가독성을 높여주는 다음 코드로 작성할 수도 있다.

${({ theme, color }) => {

const selected = theme.palette[color];

return css`

background: ${selected};

&:hover {

background: ${lighten(0.1, selected)};

}

&:active {

background: ${darken(0.1, selected)};

}

`;

}}

또는 다음과 같이 색상에 관련된 코드를 분리하여 사용할 수도 있다.

import React from 'react';

import styled, { css } from 'styled-components';

import { darken, lighten } from 'polished';

const colorStyles = css`

${({ theme, color }) => {

const selected = theme.palette[color];

return css`

background: ${selected};

&:hover {

background: ${lighten(0.1, selected)};

}

&:active {

background: ${darken(0.1, selected)};

}

`;

}}

`;

const StyledButton = styled.button`

/\* 공통 스타일 \*/

display: inline-flex;

outline: none;

border: none;

border-radius: 4px;

color: white;

font-weight: bold;

cursor: pointer;

padding-left: 1rem;

padding-right: 1rem;

/\* 크기 \*/

height: 2.25rem;

font-size: 1rem;

/\* 색상 \*/

${colorStyles}

/\* 기타 \*/

& + & {

margin-left: 1rem;

}

`;

function Button({ children, color, ...rest }) {

return <StyledButton color={color} {...rest}>{children}</StyledButton>;

}

Button.defaultProps = {

color: 'blue'

};

export default Button;

이전과 동일하게 작동한다.

이제 사이즈도 커스터마이징 해보도록 하자.

button.js

import React from 'react';

import styled, { css } from 'styled-components';

import { darken, lighten } from 'polished';

const colorStyles = css`

  ${({ theme, color }) => {

    const selected = theme.palette[color];

    return css`

      background: ${selected};

      &:hover {

        background: ${lighten(0.1, selected)};

      }

      &:active {

        background: ${darken(0.1, selected)};

      }

    `;

  }}

`;

const sizeStyles = css`

  ${props =>

    props.size === 'large' &&

    css`

      height: 3rem;

      font-size: 1.25rem;

    `}

  ${props =>

    props.size === 'medium' &&

    css`

      height: 2.25rem;

      font-size: 1rem;

    `}

    ${props =>

      props.size === 'small' &&

      css`

        height: 1.75rem;

        font-size: 0.875rem;

      `}

`;

const StyledButton = styled.button`

  /\* 공통 스타일 \*/

  display: inline-flex;

  outline: none;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  padding-left: 1rem;

  padding-right: 1rem;

  /\* 크기 \*/

  ${sizeStyles}

  /\* 색상 \*/

  ${colorStyles}

  /\* 기타 \*/

  & + & {

    margin-left: 1rem;

  }

`;

function Button({ children, color, size,  ...rest }) {

  return (

    <StyledButton color={color} size={size} {...rest}>

      {children}

    </StyledButton>

  );

}

Button.defaultProps = {

  color: 'blue'

};

export default Button;

App.js

import React from 'react';

import styled, { ThemeProvider } from 'styled-components';

import Button from './components/button';

const AppBlock = styled.div`

  width: 512px;

  margin: 0 auto;

  margin-top: 4rem;

  border: 1px solid black;

  padding: 1rem;

`;

const ButtonGroup = styled.div`

  & + & {

    margin-top: 1rem;

  }

`;

function App() {

  return (

    <ThemeProvider

      theme={{

        palette: {

          blue: '#228be6',

          gray: '#495057',

          pink: '#f06595'

        }

      }}

    >

      <AppBlock>

        <ButtonGroup>

          <Button size="large">BUTTON</Button>

          <Button size="medium">BUTTON</Button>

          <Button size="small">BUTTON</Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="gray" size="large">

            BUTTON

          </Button>

          <Button color="gray" size="medium">

            BUTTON</Button>

          <Button color="gray" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="pink" size="large">

            BUTTON

          </Button>

          <Button color="pink" size="medium">

            BUTTON</Button>

          <Button color="pink" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

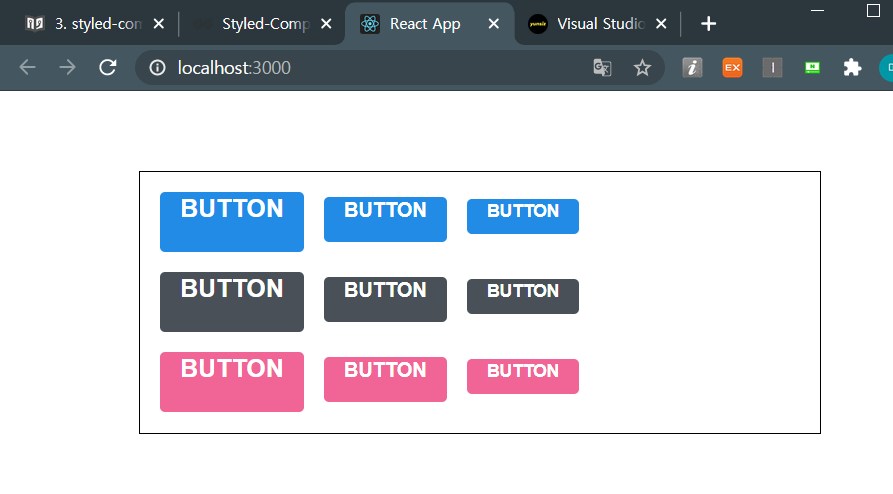
      </AppBlock>

    </ThemeProvider>

  );

}

export default App;



이와 같은 결과를 얻을 수 있는데 이상하게 블로그에는 button에 medium을 직접 설정안해줘도 되던데, 나는 medium을 설정해야만 이와 같은 결과를 얻을 수 있었다.

위 button.js는 중복되는 코드가 많으므로 다음과 같이 리팩토링하자.

import React from 'react';

import styled, { css } from 'styled-components';

import { darken, lighten } from 'polished';

const colorStyles = css`

  ${({ theme, color }) => {

    const selected = theme.palette[color];

    return css`

      background: ${selected};

      &:hover {

        background: ${lighten(0.1, selected)};

      }

      &:active {

        background: ${darken(0.1, selected)};

      }

    `;

  }}

`;

const sizes={

    large:{

        height:'3rem',

        fontSize:'1.25rem'

    },

    medium:{

        height:'2.25rem',

        fontSize:'1rem'

    },

    small:{

        height:'1.75rem',

        fontSize:'0.875rem'

    }

};

const sizeStyles = css`

  ${({size})=> css`

      height: ${sizes[size].height};

      font-size: ${sizes[size].fontSize};

    `}

`;

const StyledButton = styled.button`

  /\* 공통 스타일 \*/

  display: inline-flex;

  outline: none;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  padding-left: 1rem;

  padding-right: 1rem;

  /\* 크기 \*/

  ${sizeStyles}

  /\* 색상 \*/

  ${colorStyles}

  /\* 기타 \*/

  & + & {

    margin-left: 1rem;

  }

`;

function Button({ children, color, size,  ...rest }) {

  return (

    <StyledButton color={color} size={size} {...rest}>

      {children}

    </StyledButton>

  );

}

Button.defaultProps = {

  color: 'blue',

  size:'medium'

};

export default Button;

아마 블로그 작성하시면서 실수가 조금 있으셨던 것 같다. defaultProps는 medium으로 설정해주고 App.js리펙토링이 아닌 button.js의 리펙토링이었던 것 같다.

이제는, button 컴포넌트에 outline이라는 props를 설정하여 이 값이 true일 때에는 테두리만 지닌 버튼을 보여주도록 설정해보자. 이 작업을 할 때는 colorStyles만 수정해주면 된다.

button.js

import React from 'react';

import styled, { css } from 'styled-components';

import { darken, lighten } from 'polished';

const colorStyles = css`

  ${({ theme, color }) => {

    const selected = theme.palette[color];

    return css`

      background: ${selected};

      &:hover {

        background: ${lighten(0.1, selected)};

      }

      &:active {

        background: ${darken(0.1, selected)};

      }

      ${props=>

        props.outline&&

        css`

            color:${selected};

            background:none;

            border:1px solid ${selected};

            &:hover{

                background:${selected};

                color:white;

            }

        `}

    `;

  }}

`;

const sizes={

    large:{

        height:'3rem',

        fontSize:'1.25rem'

    },

    medium:{

        height:'2.25rem',

        fontSize:'1rem'

    },

    small:{

        height:'1.75rem',

        fontSize:'0.875rem'

    }

};

const sizeStyles = css`

  ${({size})=> css`

      height: ${sizes[size].height};

      font-size: ${sizes[size].fontSize};

    `}

`;

const StyledButton = styled.button`

  /\* 공통 스타일 \*/

  display: inline-flex;

  outline: none;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  padding-left: 1rem;

  padding-right: 1rem;

  /\* 크기 \*/

  ${sizeStyles}

  /\* 색상 \*/

  ${colorStyles}

  /\* 기타 \*/

  & + & {

    margin-left: 1rem;

  }

`;

function Button({ children, color, size, outline,  ...rest }) {

  return (

    <StyledButton color={color} size={size} outline={outline}

    {...rest}>{children}

    </StyledButton>

  );

}

Button.defaultProps = {

  color: 'blue',

  size:'medium'

};

export default Button;

App.js

import React from 'react';

import styled, { ThemeProvider } from 'styled-components';

import Button from './components/button';

const AppBlock = styled.div`

  width: 512px;

  margin: 0 auto;

  margin-top: 4rem;

  border: 1px solid black;

  padding: 1rem;

`;

const ButtonGroup = styled.div`

  & + & {

    margin-top: 1rem;

  }

`;

function App() {

  return (

    <ThemeProvider

      theme={{

        palette: {

          blue: '#228be6',

          gray: '#495057',

          pink: '#f06595'

        }

      }}

    >

      <AppBlock>

        <ButtonGroup>

          <Button size="large">BUTTON</Button>

          <Button>BUTTON</Button>

          <Button size="small">BUTTON</Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="gray" size="large">

            BUTTON

          </Button>

          <Button color="gray">

            BUTTON</Button>

          <Button color="gray" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="pink" size="large">

            BUTTON

          </Button>

          <Button color="pink">

            BUTTON</Button>

          <Button color="pink" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

      </AppBlock>

    </ThemeProvider>

  );

}

export default App;

이제 fullWidth라는 props가 주어졌다면 버튼의 크기가 100%를 차지하도록 만들어보자.

button.js

import React from 'react';

import styled, { css } from 'styled-components';

import { darken, lighten } from 'polished';

const colorStyles = css`

  ${({ theme, color }) => {

    const selected = theme.palette[color];

    return css`

      background: ${selected};

      &:hover {

        background: ${lighten(0.1, selected)};

      }

      &:active {

        background: ${darken(0.1, selected)};

      }

      ${props=>

        props.outline&&

        css`

            color:${selected};

            background:none;

            border:1px solid ${selected};

            &:hover{

                background:${selected};

                color:white;

            }

        `}

    `;

  }}

`;

const sizes={

    large:{

        height:'3rem',

        fontSize:'1.25rem'

    },

    medium:{

        height:'2.25rem',

        fontSize:'1rem'

    },

    small:{

        height:'1.75rem',

        fontSize:'0.875rem'

    }

};

const sizeStyles = css`

  ${({size})=> css`

      height: ${sizes[size].height};

      font-size: ${sizes[size].fontSize};

    `}

`;

const fullWidthStyle=css`

    ${props=>

        props.fullWidth&&

        css`

            width:100%;

            justify-content:center;

            &+&{

                margin-left:0;

                margin-top:1rem;

            }

        `

    }

`;

const StyledButton = styled.button`

  /\* 공통 스타일 \*/

  display: inline-flex;

  outline: none;

  border: none;

  border-radius: 4px;

  color: white;

  font-weight: bold;

  cursor: pointer;

  padding-left: 1rem;

  padding-right: 1rem;

  /\* 크기 \*/

  ${sizeStyles}

  /\* 색상 \*/

  ${colorStyles}

  /\* 기타 \*/

  & + & {

    margin-left: 1rem;

  }

  ${fullWidthStyle}

`;

function Button({ children, color, size, outline, fullWidth,  ...rest }) {

  return (

    <StyledButton color={color} size={size} outline={outline}

    fullWidth={fullWidth}{...rest}>{children}

    </StyledButton>

  );

}

Button.defaultProps = {

  color: 'blue',

  size:'medium'

};

export default Button;

여기서 fullWidth 스타일을 지닌 컴포넌트들도 렌더링해보자

이때 BUTTON의 옆 공백을 설정해주었음에도 불구하고 BUTTON들이 붙어서 나오는데

fullWidth의 &+& 부분을 &:not(:first-child)라고 바꿔주자. 댓글에서 나온 해결책인데 예전에는 적용되던게 버전이 바뀌고 나서 안되나보다. 이렇게 바꿔주어도 해결되진 않았지만 전보단 나은듯하다.

App.js를 다음과 같이 수정하자

import React from 'react';

import styled, { ThemeProvider } from 'styled-components';

import Button from './components/button';

const AppBlock = styled.div`

  width: 512px;

  margin: 0 auto;

  margin-top: 4rem;

  border: 1px solid black;

  padding: 1rem;

`;

const ButtonGroup = styled.div`

  & + & {

    margin-top: 1rem;

  }

`;

function App() {

  return (

    <ThemeProvider

      theme={{

        palette: {

          blue: '#228be6',

          gray: '#495057',

          pink: '#f06595'

        }

      }}

    >

      <AppBlock>

        <ButtonGroup>

          <Button size="large">BUTTON</Button>

          <Button>BUTTON</Button>

          <Button size="small">BUTTON</Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="gray" size="large">

            BUTTON

          </Button>

          <Button color="gray">BUTTON</Button>

          <Button color="gray" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="pink" size="large">

            BUTTON

          </Button>

          <Button color="pink">BUTTON</Button>

          <Button color="pink" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button size="large" outline>

            BUTTON

          </Button>

          <Button color="gray" outline>

            BUTTON

          </Button>

          <Button color="pink" size="small" outline>

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button size="large" fullWidth>

            BUTTON

          </Button>

          <Button size="large" color="gray" fullWidth>

            BUTTON

          </Button>

          <Button size="large" color="pink" fullWidth>

            BUTTON

          </Button>

        </ButtonGroup>

      </AppBlock>

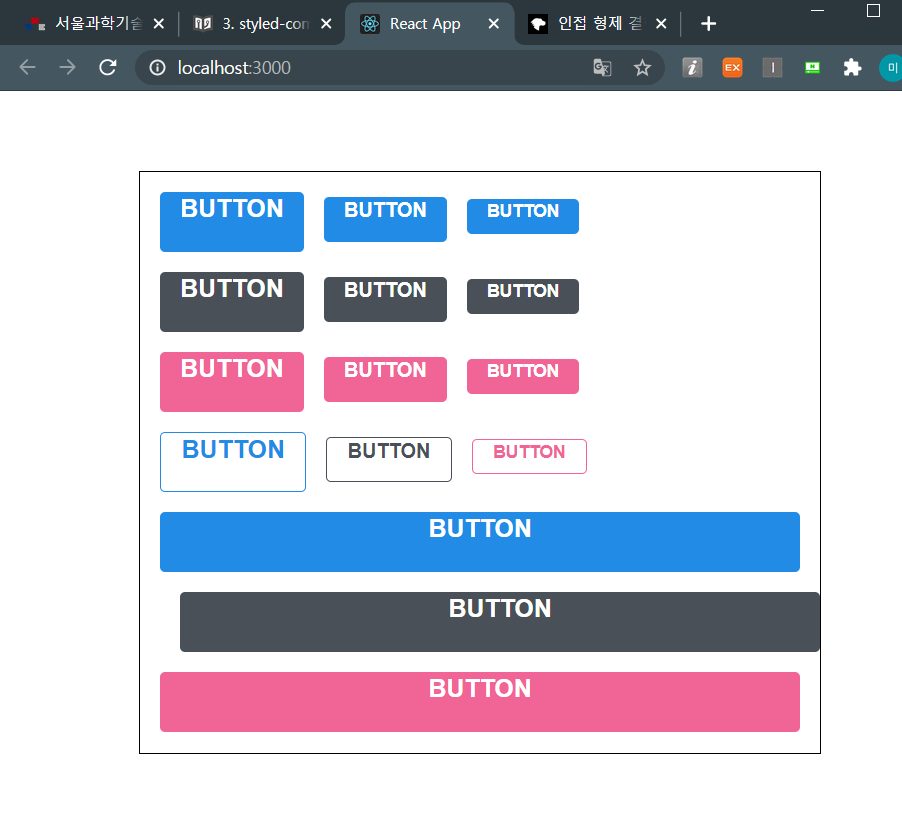
    </ThemeProvider>

  );

}

export default App;

그러면



이런 결과를 얻을 수 있다. 정렬이 안되어있는게 앞서 이야기했던 문제이다. 지금은 해결책을 모르겠으니 넘어가도록 하겠다.

**Dialog 만들기**

이번에는 기존 화면을 가리게 되면서 정보를 보여주는 Dialog 컴포넌트를 만들어보자 components 디렉터리에 Dialog.js 파일을 생성 후 다음 코드를 입력하자.

import React from 'react';

import styled from 'styled-components';

import Button from './button';

const DarkBackground=styled.div`

    position:fixed;

    left:0;

    top:0;

    width:100%;

    height:100%;

    display:flex;

    align-items:center;

    justify-content:center;

    background:rgba(0,0,0,0.8);

`;

const DialogBlock=styled.div`

    width:320px;

    padding:1.5rem;

    background:white;

    border-radius:2px;

    h3{

        margin:0;

        font-size:1.5rem;

    }

    p{

        font-size:1.125rem;

    }

`;

const ButtonGroup=styled.div`

    margin-top:3rem;

    display:flex;

    justify-content:flex-end;

`;

function Dialog({title,children,confirmText,cancelText}){

    return(

        <DarkBackground>

            <DialogBlock>

                <h3>{title}</h3>

                <p>{children}</p>

                <ButtonGroup>

                    <Button color="gray">{cancelText}</Button>

                    <Button color="pink">{confirmText}</BUtton>

                </ButtonGroup>

            </DialogBlock>

        </DarkBackground>

    );

}

Dialog.defaultProps={

    confirmText:'확인',

    cancelText:'취소'

};

export default Dialog;

h3과 p를 스타일링 때 굳이 따로 컴포넌트를 만들어주지 않아도 styled-components에서 Nested CSS 문법을 사용할 수 있기 때문에 DialogBlock안에 있는 h3와 p에게 특정 스타일을 줄 수 있다.

App.js

import React from 'react';

import styled from 'styled-components';

import Button from './button';

const DarkBackground=styled.div`

    position:fixed;

    left:0;

    top:0;

    width:100%;

    height:100%;

    display:flex;

    align-items:center;

    justify-content:center;

    background:rgba(0,0,0,0.8);

`;

const DialogBlock=styled.div`

    width:320px;

    padding:1.5rem;

    background:white;

    border-radius:2px;

    h3{

        margin:0;

        font-size:1.5rem;

    }

    p{

        font-size:1.125rem;

    }

`;

const ButtonGroup=styled.div`

    margin-top:3rem;

    display:flex;

    justify-content:flex-end;

`;

function Dialog({title,children,confirmText,cancelText}){

    return(

        <DarkBackground>

            <DialogBlock>

                <h3>{title}</h3>

                <p>{children}</p>

                <ButtonGroup>

                    <Button color="gray">{cancelText}</Button>

                    <Button color="pink">{confirmText}</Button>

                </ButtonGroup>

            </DialogBlock>

        </DarkBackground>

    );

}

Dialog.defaultProps={

    confirmText:'확인',

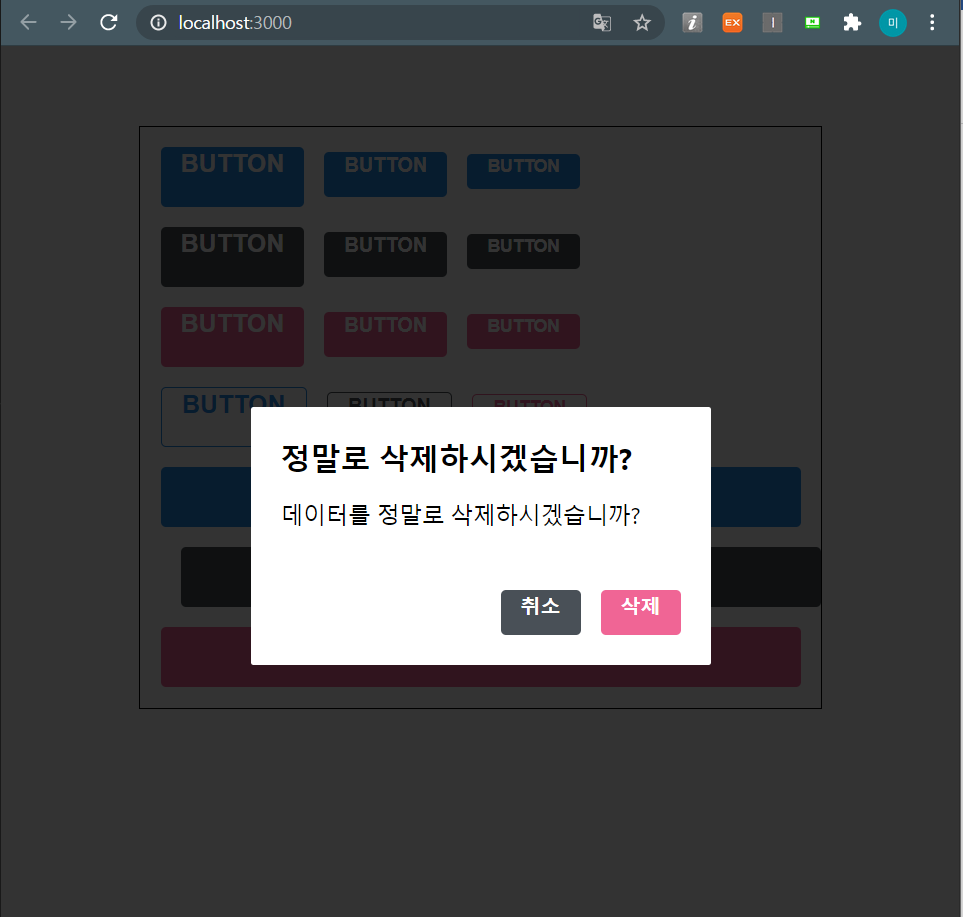
    cancelText:'취소'

};

export default Dialog;

Dialog 컴포넌트를 예시 내용과 함께 AppBlock 하단에 넣어주었으며 ThemeProvider 내부는 하나의 리액트 엘리먼트로 감싸져있어야하기 때문에 AppBlock과 Dialog를 <></>으로 감싸주었다.

아래를 보면 Dialog에서 취소 버튼과 삭제 버튼의 간격이 조금 넓어 보이는 느낌이 있는데, styled-components로 컴포넌트의 스타일을 특정 상황에서 덮어쓰는 방법에 대해서 알아보자.



Dialog.js에서 다음과 같이 ShortMarginButton을 만들고 기존 Button을 대체시켜보자.

const ShortMarginButton=styled(Button)`

    &+&{

        margin-left:0.5rem;

    }

`;

function Dialog({title,children,confirmText,cancelText}){

    return(

        <DarkBackground>

            <DialogBlock>

                <h3>{title}</h3>

                <p>{children}</p>

                <ButtonGroup>

                    <ShortMarginButton color="gray">{cancelText}</ShortMarginButton>

                    <ShortMarginButton color="pink">{confirmText}</ShortMarginButton>

                </ButtonGroup>

            </DialogBlock>

        </DarkBackground>

    );

}

그러면 여백이 줄어듦을 확인할 수 있다. 이렇게 컴포넌트의 스타일을 커스터마이징할 때 에는 해당 컴포넌트에서 className props를 내부 엘리먼트에게 전달이 되고 있는지 확인해주어야 합니다.

const MyComponent = ({ className }) => {

return <div className={className}></div>

};

const ExtendedComponent = styled(MyComponent)`

background: black;

`;

참고로 우리가 만든 Button 컴포넌트의 경우에는 *…rest* 를 통하여 전달이 되고 있다. 컴포넌트의 모양새를 모두 갖추었으면 열고 닫을 수 있는 기능을 구현해보자. Dialog에서 onConfirm과 onCancel을 props로 받아오도록 하고 해당 함수들을 각 버튼들에게 onClick으로 설정해주자.

그리고, visible props도 받아와서 이 값이 false 일 때 컴포넌트에서 null을 반환하도록 설정해주자.

import React from 'react';

import styled from 'styled-components';

import Button from './button';

const DarkBackground=styled.div`

    position:fixed;

    left:0;

    top:0;

    width:100%;

    height:100%;

    display:flex;

    align-items:center;

    justify-content:center;

    background:rgba(0,0,0,0.8);

`;

const DialogBlock=styled.div`

    width:320px;

    padding:1.5rem;

    background:white;

    border-radius:2px;

    h3{

        margin:0;

        font-size:1.5rem;

    }

    p{

        font-size:1.125rem;

    }

`;

const ButtonGroup=styled.div`

    margin-top:3rem;

    display:flex;

    justify-content:flex-end;

`;

const ShortMarginButton=styled(Button)`

    &+&{

        margin-left:0.5rem;

    }

`;

function Dialog({title,children,confirmText,cancelText,onConfirm,onCancel,visible})

{

    if(!visible) return null;

    return(

        <DarkBackground>

            <DialogBlock>

                <h3>{title}</h3>

                <p>{children}</p>

                <ButtonGroup>

                    <ShortMarginButton color="gray" onClick={onCancel}>{cancelText}</ShortMarginButton>

                    <ShortMarginButton color="pink" onClick={onConfirm}>{confirmText}</ShortMarginButton>

                </ButtonGroup>

            </DialogBlock>

        </DarkBackground>

    );

}

Dialog.defaultProps={

    confirmText:'확인',

    cancelText:'취소'

};

export default Dialog;

그 다음에는 App컴포넌트에서 useState를 사용하여 Dialog를 가시성 상태를 관리하자.

App.js

import React, {useState} from 'react';

import styled, { ThemeProvider } from 'styled-components';

import Button from './components/button';

import Dialog from './components/Dialog';

const AppBlock = styled.div`

  width: 512px;

  margin: 0 auto;

  margin-top: 4rem;

  border: 1px solid black;

  padding: 1rem;

`;

const ButtonGroup = styled.div`

  & + & {

    margin-top: 1rem;

  }

`;

function App() {

  const [dialog,setDialog]=useState(false);

  const onClick=()=>{

    setDialog(true);

  };

  const onConfirm=()=>{

    console.log('확인');

    setDialog(false);

  }

  const onCancel=()=>{

    console.log('취소');

    setDialog(false);

  }

  return (

    <ThemeProvider

      theme={{

        palette: {

          blue: '#228be6',

          gray: '#495057',

          pink: '#f06595'

        }

      }}

    >

      <>

      <AppBlock>

        <ButtonGroup>

          <Button size="large">BUTTON</Button>

          <Button>BUTTON</Button>

          <Button size="small">BUTTON</Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="gray" size="large">

            BUTTON

          </Button>

          <Button color="gray">BUTTON</Button>

          <Button color="gray" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button color="pink" size="large">

            BUTTON

          </Button>

          <Button color="pink">BUTTON</Button>

          <Button color="pink" size="small">

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button size="large" outline>

            BUTTON

          </Button>

          <Button color="gray" outline>

            BUTTON

          </Button>

          <Button color="pink" size="small" outline>

            BUTTON

          </Button>

        </ButtonGroup>

        <ButtonGroup>

          <Button size="large" fullWidth>

            BUTTON

          </Button>

          <Button size="large" color="gray" fullWidth>

            BUTTON

          </Button>

          <Button size="large" color="pink" fullWidth onClick={onClick}>

            삭제

          </Button>

        </ButtonGroup>

      </AppBlock>

      <Dialog

        title="정말로 삭제하시겠습니까?"

        confirmText="삭제"

        cancelText="취소"

        onConfirm={onConfirm}

        onCancel={onCancel}

        visible={dialog}

        >

          데이터를 정말로 삭제하시겠습니까?

        </Dialog>

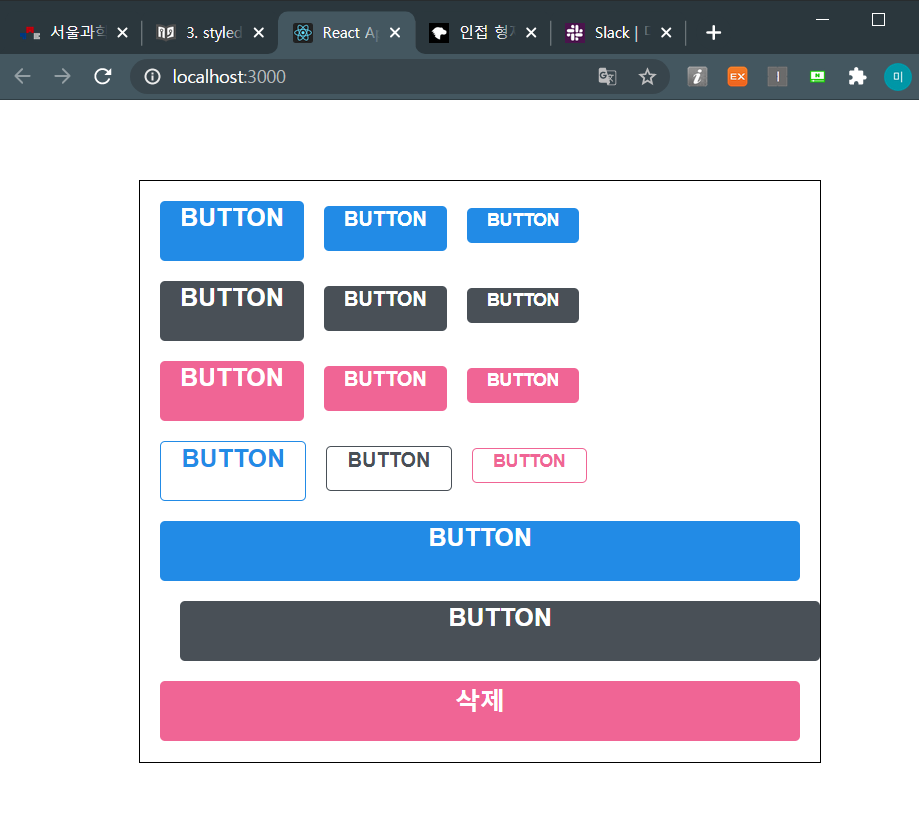
        </>

    </ThemeProvider>

  );

}

export default App;



맨 아래에 있는 fullWidth 핑크색 버튼의 이름을 삭제로 변경한 후 해당 버튼을 누르면 우리가 만든 Dialog가 보여지도록 설정을 했고, Dialog에 onConfirm, onoCancel, visible 값을 전달해주었다.

**트랜지션 구현하기**

이제 Dialog의 기능은 모두 구현해주었다. 이번에는 Dialog가 나타나거나 사라질 때 트랜지션 효과를 적용해보자. 트랜지션 효과를 적용할때에는 CSS Keyframe을 사용하며, styled-components에서 이를 사용할 때에는 *keyframes* 라는 유틸을 사용한다.

Dialog가 나타날 때 DarkBackground 쪽에는 서서히 나타나는 fadeln 효과를 주고, DialogBlcok에는 아래에서부터 위로 올라오는 효과를 보여주는 slideUp효과를 줘보자. 애니메이션의 이름은 마음대로 지정할 수 있다.

Dialog.js

import React from 'react';

import styled,{keyframes} from 'styled-components';

import Button from './button';

const fadeIn=keyframes`

    from{

        opacity:0

    }

    to{

        opacity:1

    }

`;

const slideUp=keyframes`

    from{

        trnasform:translateY(200px);

    }

    to{

        transform:translateY(0px);

    }

`;

const DarkBackground=styled.div`

    position:fixed;

    left:0;

    top:0;

    width:100%;

    height:100%;

    display:flex;

    align-items:center;

    justify-content:center;

    background:rgba(0,0,0,0.8);

    animation-duration:0.25s;

    animation-timing-functino:ease-out;

    animation-name:${fadeIn};

    animation-fill-mode:forwards;

`;

const DialogBlock=styled.div`

    width:320px;

    padding:1.5rem;

    background:white;

    border-radius:2px;

    h3{

        margin:0;

        font-size:1.5rem;

    }

    p{

        font-size:1.125rem;

    }

    animation-duration:0.25s;

    animation-timing-function:ease-out;

    animation-name:${slideUp};

    animation-fill-mode:forwards;

`;

const ButtonGroup=styled.div`

    margin-top:3rem;

    display:flex;

    justify-content:flex-end;

`;

const ShortMarginButton=styled(Button)`

    &+&{

        margin-left:0.5rem;

    }

`;

function Dialog({title,children,confirmText,cancelText,onConfirm,onCancel,visible})

{

    if(!visible) return null;

    return(

        <DarkBackground>

            <DialogBlock>

                <h3>{title}</h3>

                <p>{children}</p>

                <ButtonGroup>

                    <ShortMarginButton color="gray" onClick={onCancel}>{cancelText}</ShortMarginButton>

                    <ShortMarginButton color="pink" onClick={onConfirm}>{confirmText}</ShortMarginButton>

                </ButtonGroup>

            </DialogBlock>

        </DarkBackground>

    );

}

Dialog.defaultProps={

    confirmText:'확인',

    cancelText:'취소'

};

export default Dialog;

이렇게 하면 컴포넌트가 나타날 때 트랜지션 효과가 나타날 것이다. 이제 사라지는 트랜지션 효과를 만들어볼텐데, 구현하기 조금 까다롭지만 원리만 알면 굉장히 쉽다.

사라지는 효과를 구현하려면 Dialog 컴포넌트에서 두개의 로컬 상태를 관리해주어야 한다. 하나는 현재 트랜지션 효과를 보여주고 있는 중이라는 상태를 의미하는 *animate*, 나머지 하나는 실제로 컴포넌트가 사라지는 시점을 지연시키기 위한 *localVisible* 값이다.

그리고 useEffect를 하나 작성해서, visible 값이 true에서 false로 바뀌는 시점을 감지하여 animate 값을 true로 바꿔주고 *setTimeout* 함수를 사용하여 250ms 이후 false로 바꾸어주어야 한다.

추가적으로 !visible 조건에서 null을 반환하는 대신에 *!animate && !localVisible 조건에서* null을 반환하도록 수정해줘야 한다.

Dialog.js

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

import styled,{keyframes} from 'styled-components';

import Button from './button';

const fadeIn=keyframes`

    from{

        opacity:0

    }

    to{

        opacity:1

    }

`;

const slideUp=keyframes`

    from{

        trnasform:translateY(200px);

    }

    to{

        transform:translateY(0px);

    }

`;

const DarkBackground=styled.div`

    position:fixed;

    left:0;

    top:0;

    width:100%;

    height:100%;

    display:flex;

    align-items:center;

    justify-content:center;

    background:rgba(0,0,0,0.8);

    animation-duration:0.25s;

    animation-timing-functino:ease-out;

    animation-name:${fadeIn};

    animation-fill-mode:forwards;

`;

const DialogBlock=styled.div`

    width:320px;

    padding:1.5rem;

    background:white;

    border-radius:2px;

    h3{

        margin:0;

        font-size:1.5rem;

    }

    p{

        font-size:1.125rem;

    }

    animation-duration:0.25s;

    animation-timing-function:ease-out;

    animation-name:${slideUp};

    animation-fill-mode:forwards;

`;

const ButtonGroup=styled.div`

    margin-top:3rem;

    display:flex;

    justify-content:flex-end;

`;

const ShortMarginButton=styled(Button)`

    &+&{

        margin-left:0.5rem;

    }

`;

function Dialog({title,children,confirmText,cancelText,onConfirm,onCancel,visible})

{

    const [animate,setAnimate]=useState(false);

    const [localVisible, setLocalVisible]=useState(visible);

    useEffect(()=>{

        if(localVisible&&!visible){

            setAnimate(true);

            setTimeout(()=>setAnimate(false),250);

        }

        setLocalVisible(visible);

    },[localVisible,visible]);

    if(!animate&&!localVisible) return null;

    return(

        <DarkBackground>

            <DialogBlock>

                <h3>{title}</h3>

                <p>{children}</p>

                <ButtonGroup>

                    <ShortMarginButton color="gray" onClick={onCancel}>{cancelText}</ShortMarginButton>

                    <ShortMarginButton color="pink" onClick={onConfirm}>{confirmText}</ShortMarginButton>

                </ButtonGroup>

            </DialogBlock>

        </DarkBackground>

    );

}

Dialog.defaultProps={

    confirmText:'확인',

    cancelText:'취소'

};

export default Dialog;

이제 확인/취소를 눌렀을 때 약간이 딜레이 이후에 Dialog가 사라지는 것을 확인 할 수 있다. 이제 DarkBackground와 DialogBlock에 *disappear*라는 props를 주어서 사라지는 효과가 나타나도록 설정을 해보자.

각 컴포넌트의 disappear값은 !visible 로 하면 된다.

Dialog.js

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

import styled, { keyframes, css } from 'styled-components';

import Button from './button';

const fadeIn = keyframes`

  from {

    opacity: 0

  }

  to {

    opacity: 1

  }

`;

const fadeOut = keyframes`

  from {

    opacity: 1

  }

  to {

    opacity: 0

  }

`;

const slideUp = keyframes`

  from {

    transform: translateY(200px);

  }

  to {

    transform: translateY(0px);

  }

`;

const slideDown = keyframes`

  from {

    transform: translateY(0px);

  }

  to {

    transform: translateY(200px);

  }

`;

const DarkBackground = styled.div`

  position: fixed;

  left: 0;

  top: 0;

  width: 100%;

  height: 100%;

  display: flex;

  align-items: center;

  justify-content: center;

  background: rgba(0, 0, 0, 0.8);

  animation-duration: 0.25s;

  animation-timing-function: ease-out;

  animation-name: ${fadeIn};

  animation-fill-mode: forwards;

  ${props =>

    props.disappear &&

    css`

      animation-name: ${fadeOut};

    `}

`;

const DialogBlock = styled.div`

  width: 320px;

  padding: 1.5rem;

  background: white;

  border-radius: 2px;

  h3 {

    margin: 0;

    font-size: 1.5rem;

  }

  p {

    font-size: 1.125rem;

  }

  animation-duration: 0.25s;

  animation-timing-function: ease-out;

  animation-name: ${slideUp};

  animation-fill-mode: forwards;

  ${props =>

    props.disappear &&

    css`

      animation-name: ${slideDown};

    `}

`;

const ButtonGroup = styled.div`

  margin-top: 3rem;

  display: flex;

  justify-content: flex-end;

`;

const ShortMarginButton = styled(Button)`

  & + & {

    margin-left: 0.5rem;

  }

`;

function Dialog({

  title,

  children,

  confirmText,

  cancelText,

  onConfirm,

  onCancel,

  visible

}) {

  const [animate, setAnimate] = useState(false);

  const [localVisible, setLocalVisible] = useState(visible);

  useEffect(() => {

    // visible 값이 true -> false 가 되는 것을 감지

    if (localVisible && !visible) {

      setAnimate(true);

      setTimeout(() => setAnimate(false), 250);

    }

    setLocalVisible(visible);

  }, [localVisible, visible]);

  if (!animate && !localVisible) return null;

  return (

    <DarkBackground disappear={!visible}>

      <DialogBlock disappear={!visible}>

        <h3>{title}</h3>

        <p>{children}</p>

        <ButtonGroup>

          <ShortMarginButton color="gray" onClick={onCancel}>

            {cancelText}

          </ShortMarginButton>

          <ShortMarginButton color="pink" onClick={onConfirm}>

            {confirmText}

          </ShortMarginButton>

        </ButtonGroup>

      </DialogBlock>

    </DarkBackground>

  );

}

Dialog.defaultProps = {

  confirmText: '확인',

  cancelText: '취소'

};

export default Dialog;

모든 트랜지션 효과를 넣었다! 끝!!