

# react 프로젝트 생성

react 프로젝트 생성은 npx를 이용한 명령어를 통해 가능합니다.

```
npx create-react-app project
```

해당 명령어를 입력하면 하위 폴더로 project 폴더가 생성되고 그 안에 리액트 프로젝트가 생성됩니다.  
package.json 안에는 자동으로 생성된 script가 있습니다. 하나하나씩 알아보겠습니다.

```
"scripts": {  
  "start": "react-scripts start", // develop 환경으로 프로젝트를 실행할 때 사용합니다.  
  "build": "react-scripts build", // 배포를 위한 build 작업을 할 때 사용됩니다.  
  "test": "react-scripts test", // test 코드를 돌릴 때 사용합니다.  
  "eject": "react-scripts eject" // 프로젝트 자체를 커스텀하기 위한 webpack, babel, eslint 등의 설정환경을 셋팅합니다.  
  // 한번 셋팅하면 다시 되돌릴수 없습니다.  
},
```

# react 프로젝트 구조

create-react-app을 통해 생성한 프로젝트는 다음의 구조를 가지고 있습니다.

```
-- node_modules  
-- public  
-- src  
  -- index.js  
  -- App.js  
.gitignore  
package-lock.json  
package.json  
README.md
```

- **node\_modules**: 프로젝트에 사용할 라이브러리가 설치된 폴더입니다.
- **public**: 정적인 파일들(이미지, 비디오 등등)이 포함된 폴더입니다.
- **src**: react의 주된 source 폴더입니다. 컴포넌트의 구현은 이 폴더안에서 합니다.
  - **index.js**: react의 진입지점입니다. App.js를 불러와서 index.html에 매칭시켜주는 역할을 합니다.
  - **App.js**: 가장 최상위 컴포넌트입니다.

# react 프로젝트 구조 추가

기존의 폴더 구조에서 3개의 폴더를 추가해봅시다.

```
-- node_modules  
-- public  
-- src  
  -- components  
  -- layouts  
  -- pages  
  -- index.js  
  -- App.js  
.gitignore  
package-lock.json  
package.json  
README.md
```

- components: UI구성요소들을 구현하는 컴포넌트들을 모아두는곳 입니다.
- layouts: page의 layout을 구성하는 컴포넌트들을 모아두는곳 입니다.
- pages: page 역할의 컴포넌트들을 모아두는곳 입니다.

리액트는 화면을 구성하는 단위가 컴포넌트 입니다. 그래서 컴포넌트의 범위에 따라 page, layout, component로 나눠볼 생각입니다.

# react 컴포넌트 생성

react의 컴포넌트 형태에는 2가지가 있습니다.

하나는 function 타입이고, 다른 하나는 class 타입입니다.

두가지 형태 다 동일한 기능을 하지만 컴포넌트의 형태에 따라 코드 구현방식이 달라집니다.

function 타입

```
function welcome(props) {  
  return <h1>Hello world</h1>  
}
```

class 타입

```
class welcome extends React.Component {  
  render() {  
    return <h1>Hello world</h1>  
  }  
}
```

function 타입의 경우 hook 함수들을 통한 컴포넌트 제어방식을 사용하며

class 타입의 경우 lifecycle 함수들을 통해 컴포넌트를 제어하게 됩니다.

# react 컴포넌트 생성

src - pages에 dashboard.jsx 파일을 만들어봅시다.

react에선 js와 html을 같이 사용하는 문법을 사용하는데 이를 jsx라 합니다.

dashboard.jsx 파일에 다음의 코드를 작성해 봅시다.

```
function Dashboard() {  
  return (  
    <h1>Hello Dashboard !!!</h1>  
  );  
}  
  
export default Dashboard;
```

코드가 상당히 간단합니다. 코드를 살펴보자면 Dashboard라는 함수는 html을 반환하는 함수이고 해당 함수를 export default 하고 있습니다.

export default 는 es6 문법의 import/export에서 해당 파일을 import 할 때 다른 명시적으로 import 시키는 항목이 없다면 기본적으로 import 되는걸 지정해주는 문법입니다.

# react 컴포넌트 생성

Dashboard 컴포넌트를 만들었으니 이제 App.js 파일을 수정해봅니다.

```
import Dashboard123 from './pages/dashboard';

function App() {
  return (
    <Dashboard123></Dashboard123>
  );
}

export default App;
```

pages의 dashboard.jsx 파일을 import 했습니다.

import 할 때 붙여주는 Dashboard123는 import 하는 명칭을 지정해주는것이기 때문에 임의로 변경해도 상관없습니다.  
다만 명칭을 헷갈리지 않게 지어주면 됩니다.

# props

react의 컴포넌트는 Tree 구조를 가지고 있습니다.

그래서 컴포넌트들은 부모 컴포넌트와 자식 컴포넌트 구조를 가지고 있습니다.

그리고 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달할 수 있는데 이런 역할을 하는것이 props 입니다.

```
function SayMyName(props) {  
  return (  
    <h1>My name is {props.name}</h1>  
  );  
}  
  
export default SayMyName;
```

위의 코드는 sayMyName 이라는 컴포넌트입니다.

함수의 매개변수로 props를 받고 있고 props의 name값을 사용하고 있습니다.

이 SayMyName 컴포넌트를 사용하는 부분을 보겠습니다.

# props

```
import SayMyName from "../components/sayMyName";

function Ex1() {
  return (
    <>
    <SayMyName name="ex1"></SayMyName>
    <SayMyName name="ex2"></SayMyName>
    <SayMyName name="ex3"></SayMyName>
  </>
);
}

export default Ex1;
```

SayMyName에 사용된 props.name 은 name 속성으로 입력된 데이터입니다.

이렇게 부모 컴포넌트에서 자식 컴포넌트로 데이터를 주입하고 싶으면 해당 컴포넌트에 속성명=값 형태로 넣으면 됩니다.

# props.children

간혹 부모 컴포넌트에서 jsx를 넘겨주고 싶은 경우가 있습니다.  
이럴경우 어떻게 주입시켜주는지 알아보겠습니다.

```
function MyLoader(props) {  
  
  const { isLoading, children } = props  
  
  return (  
    isLoading ? <h1>...Loading</h1> : children  
  );  
}  
  
export default MyLoader;
```

MyLoader 컴포넌트는 isLoading, children 을 입력받습니다.  
isLoading값에 따라서 true 일 땐 ‘...Loading’ 을 보여주고 false일 땐 props.children을 보여주게 했습니다.

이제 MyLoader를 사용하는 컴포넌트를 확인해보겠습니다.

## props.children

Ex2 컴포넌트에서 MyLoader 컴포넌트를 사용하는 코드입니다.

```
import MyLoader from "../components/loader";

function Ex2() {
  return (
    <>
    <MyLoader isLoading={true}>
      <p>load complete !!!</p>
    </MyLoader>
    <MyLoader>
      <p>load complete !!!</p>
    </MyLoader>
  </>
  );
}

export default Ex2;
```

위의 MyLoader는 isLoading값이 true로 로딩상태를 나타내주게 됩니다. 하지만 밑의 MyLoader는 isLoading값이 없기 때문에 로딩상태를 표시하지 않고 안의 내용으로 들어간 'load complete !!!' 를 표시하게 됩니다.  
이렇게 부모 컴포넌트에서 자식 컴포넌트로 jsx를 주입하고 싶을 땐 props.children을 이용해서 전달해줄 수 있습니다.

# event

react에서 event binding을 알아보겠습니다.

react의 jsx에서 dom 속성에 접근할 땐 camelCase 가 사용됩니다.

우선 click 이벤트에 접근하는 코드를 보겠습니다.

```
import React from 'react'

function EventFunc(props) {

  function clickHandler() {
    alert(`clicked !!!`)
  }

  return (
    <div>
      <button onClick={clickHandler}>Click Bind</button>
      <button onClick={() => clickHandler()}>Click Arrow</button>
    </div>
  )
}

export default EventFunc
```

# event

- onClick: onclick을 camelCase로 적어야 해당 이벤트에 접근할 수 있습니다. 다른 이벤트 또한 같은 방식으로 명칭을 적어줘야 해당 이벤트에 접근할 수 있습니다.
- binding: clickHandler 를 바인딩 해주고 있습니다. 함수를 그냥 불러서 바인딩 시켜줄 수 있지만 arrow-function 을 통해서 바인딩 시켜줄 수 있습니다. event 객체를 전달하거나 매개변수를 전달해야 할 경우 arrow-function을 사용합니다.

## event - class

class 형식의 컴포넌트에서 이벤트를 바인딩해주는 경우 this 이슈가 있습니다.

```
export class EventClass extends Component {
  constructor(props) {
    super(props)
  }

  clickHandler() {
    alert(`clicked !!`)
  }

  render() {
    return (
      <div>
        {/* <button onClick={this.clickHandler}>Click Not Work</button> */}
        <button onClick={this.clickHandler.bind(this)}>Click Bind</button>
        <button onClick={() => this.clickHandler()}>Click Arrow</button>
      </div>
    )
  }
}
```

## **event - class**

클래스 컴포넌트에서 이벤트 바인딩시 함수를 넘겨주고 함수 내에서 this를 사용하는 경우 this가 해당 함수 자체를 바라보기 때문에 클래스 컴포넌트 변수에 접근할 수 없습니다.

이럴경우 bind(this) 를 통해 해당 이슈를 해결하거나 arrow-function 을 사용해서 해결할 수 있습니다.

## EX - props & event

### AlertComp 만들기

AlertComp 를 만들어주세요. 클래스형과 함수형 두가지를 만드시면 됩니다.

각각 명칭은 AlertCompClass, AlertCompFunc 입니다.

두 컴포넌트는 props로 message를 받습니다.

컴포넌트 안에는 AlertPropsMessage 버튼이 있고 버튼을 누르면 props로 전달받은 message를 포함하여 `this is props message :: \${props.message}` 라는 문구를 띄웁니다.

함수명은 자유롭게 정하시면 됩니다.

# state

react의 컴포넌트에서 사용하는 변수 중 state가 있습니다.

state는 컴포넌트내의 변수로 다른 일반적인 변수들과는 다릅니다.

다음은 props와 state의 차이점을 적어놓은 것 입니다.

- props는 부모에게서 전달받은 값입니다.
- props는 불변입니다.
- 컴포넌트의 형태에 따라 다음방법으로 사용됩니다.
  - props - functional Component
  - this.props - Class Component
- state는 컴포넌트 안에서 사용되는 변수입니다.
- state는 값을 변화시킬 수 있습니다.
- 컴포넌트의 형태에 따라 다음 방법으로 사용됩니다.
  - useState 를 이용한 Hook 접근 - functional Component
  - this.state - Class Component

## state - setState

state의 변화는 setState함수에 의해서만 변경시킬수 있습니다.

왜냐하면 state값의 변화는 render를 동작시키며 화면을 전환시키기 때문입니다.

다음은 Class 형태의 컴포넌트에서 state를 사용하는 예시 입니다.

```
class CounterClass extends Component {  
  
  constructor() {  
    super()  
    this.state = { count: 0 }  
  }  
  
  increase() {  
    this.setState({ count: this.state.count + 1 })  
  }  
  
  render() {  
    return (  
      <div>  
        <div>count: {this.state.count}</div>  
        <button onClick={() => this.increase()}>increase</button>  
      </div>  
    )  
  }  
}
```

# state - setState

- constructor
  - constructor에서 this.state를 통해 state에 접근하고 있습니다.
  - 해당 코드는 state에 count라는 변수를 선언해주고 초기값을 0으로 설정해주는 과정입니다.
- increase 함수
  - increase 함수에서 setState를 통해 state의 값을 변경해주고 있습니다.
  - state의 값을 변경하려면 setState를 통해 변경해야 합니다.
- render 함수
  - render 함수 안에서 this.state.count를 통해 count 값에 접근하고 있습니다.
  - button onClick은 jsx에서 click 이벤트에 접근할 때 사용하는 속성값입니다.
    - jsx에서 이벤트에 접근할 경우 camelCase 방식으로 접근해야 합니다.

## state - setState

setState함수는 다음 두가지 형태로 사용 가능합니다.

- `setState(objectForState, callback?)`
  - objectForState 는 state를 변경시켜줄 object를 말합니다. object는 state의 타입과 일치해야 합니다.
  - 두번째 인자로 받는 callback 함수는 setState 동작 완료 후 구동되는 함수를 넣을 수 있습니다.
- `setState(callback, callback?)`
  - 첫번째 인자로 받는 callback 함수는 state 타입의 object를 반환하는 함수입니다. 해당 함수의 매개변수로 두개의 값이 들어오는데 하나는 현재 상태의 state 값 그리고 다른 하나는 props 값입니다.
  - 두번째 인자로 받는 callback 함수는 setState 동작 완료 후 구동되는 함수를 넣을 수 있습니다.

```
increaseSync() {
  // setState의 이전 결과를 받기위해선 callback function을 사용해야 함
  this.setState(
    (prevState, props) => { return { count: prevState.count + 1 } },
    () => { console.log('Callback value :: ', this.state.count) }
  )
}
```

## state - useState

Class 형식의 컴포넌트에선 this를 통해 state에 접근할 수 있었습니다.

그렇다면 함수형 컴포넌트에선 어떻게 state에 접근할 수 있을까요?

react에선 컴포넌트의 기능들에 접근하기 위해서 각종 Hook 함수들을 제공합니다. 그 중 state에 접근하기 위해 사용되는 Hook이 useState입니다.

사용법은 다음과 같습니다.

```
const [count, setCount] = useState(0)
```

- useState(0)
  - useState함수의 인자값으로 초기값으로 사용할 데이터를 넣어줍니다.
- count
  - useState에 넣어준 초기값으로 셋팅된 변수입니다. 클래스 컴포넌트의 this.state.count와 같습니다.
- setCount
  - useState가 생성한 state 변경 함수입니다. 클래스 컴포넌트의 this.setState와 같습니다.

## state - useState

앞에서 만든 CounterClase를 함수형으로 변경하면 다음과 같은 코드가 됩니다.

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

function Counter() {

  const [count, setCount] = useState(0)

  function increase() {
    setCount(count + 1)
  }

  return (
    <div>
      <div>count: {count}</div>
      <button onClick={() => increase()}>increase</button>
    </div>
  );
}

export default Counter;
```

## state - object 형태의 state

state가 object 형태일 경우 setState시 우리가 원하는 값만 setState에 넣어줄 경우 다른값들이 사라질 수 있습니다.  
그래서 setState 사용시 이전 state의 값을 같이 넣어줘야 하는데, 이 때 spread 문법을 사용하면 편합니다.

```
function CounterThree() {  
  
  const [name, setName] = useState({ firstName: "", lastName: "" })  
  
  return (  
    <form>  
      <input type="text" value={name.firstName} onChange={e => setName({ ...name, firstName: e.target.value })} />  
      <input type="text" value={name.lastName} onChange={e => setName({ ...name, lastName: e.target.value })} />  
      <h2>first name is : {name.firstName}</h2>  
      <h2>last name is : {name.lastName}</h2>  
      <h2>{JSON.stringify(name)}</h2>  
    </form>  
  )  
}  
  
export default CounterThree
```

## EX - state

### Counter function 컴포넌트 수정하기

앞에서 만든 counterFunc에 다음을 추가해주세요

1. diff state 를 만들어주시고 jsx에 input을 추가해서 diff값을 변경할 수 있게 바꿔주세요
2. increase 함수가 diff 값 만큼 숫자를 증가시키게 해주세요
3. decrease 함수를 만들어주세요

# List Render

react에서 list를 렌더링 하는걸 배워보겠습니다.

```
import postDummyList from '../mock/postData.json'

function Ex7() {

  const [postList, setPostList] = useState(postDummyList)

  return (
    <div>
      {
        postList.map(post => {
          return (
            <p key={post.id}>
              {post.id} : {post.title}
            </p>
          )
        })
      }
    </div>
  )
}
```

# List Render

postList를 map 함수를 통해 p 태그를 만들어서 반환시키는 로직입니다.

여기서 key라는 속성이 보이는데, 이 속성은 react에서 list를 랜더링 할 때 필수로 들어가는 값입니다.

- 1: item1
  - 2: item2
- 
- 1: item1
  - 2: item2
  - 3: item3

위와같이 리스트에 3번 아이템이 추가되었다고 가정해봅시다.

그럼 전체 리스트를 랜더링 하는 것 보다, 기존에 있는 리스트는 그대로 두고 3번 아이템이 들어가는 부분만 재 랜더링 해서 보여주는것이 해당 페이지를 처리하는과정에서 더 빠를것입니다.

이러한 랜더링 최적화를 위해 react에서는 key라는 속성을 사용하고, 이런 목록들을 만들경우 key 속성이 필수가 됩니다.

## EX - list render

### filter select 수정하기

지금은 select의 option 부분이 빠져있습니다.

option 부분에 userId 별로 option 을 구현해주세요