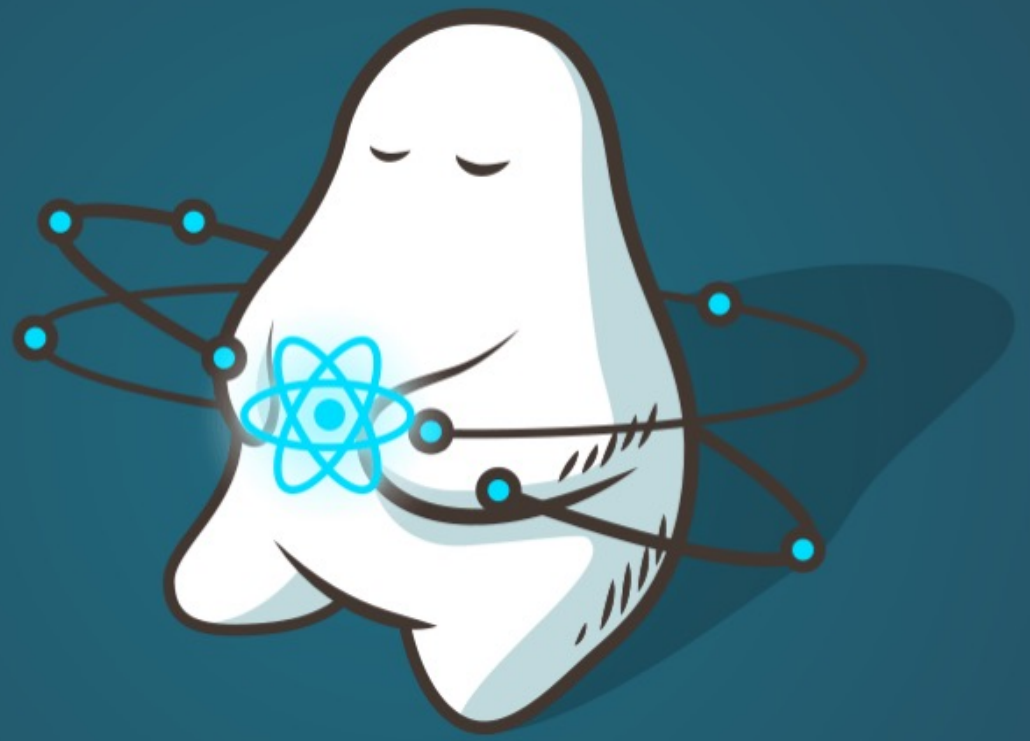





1.1 SPA

1. SPA?

- Single Page Application! 한 개의 페이지로 이루어진 앱
- MPA(Multi Page Application)과 상반된 개념

- 왜 SPA가 좋은가?
  - MPA는 어떠한 이벤트가 발생할 때마다 웹 페이지가 Reload 되어 Re-Rendering된다. (서버 사이드 랜더링)
  - 이는 데이터를 많이 잡아먹는 문제가 있다.
  - 하지만, SPA를 사용하면 필요한 부분만 수정해 웹 페이지가 Reload되는 문제가 없다. (클라이언트 사이드 랜더링)

- SSR - Server Side Rendering
  - 서버 쪽에서 렌더링 준비를 끝마친 상태로 클라이언트에 전달됨
- CSR - Client Side Rendering
  - 렌더링이 클라이언트 쪽에서 일어남

2. SPA의 장단점

SPA의 장점

- 딱 한 개의 페이지로 구성된 애플리케이션 (Augular, Vue와 비슷한 기술)
- 서버에 1회 리소스를 요청
- 필요한 데이터만 받아와 새로고침 없이 데이터 수정이 가능
- 자연스러운 UX 구현이 가능
- React를 이용하면 안드로이드/iOS 모바일 앱까지 제작 가능 (Expo)

SPA의 단점

- 싱글 페이지이기 때문에 SEO에 취약 -> 이를 보안하기 위해 NextJS가 등장

3. 리액트?

- 2013년에 페이스북에서 발표한 오픈소스 자바스크립트 프레임워크
- 가상 DOM(Document object model)과 JSX(JavaScript XML) 방식으로 동작하는 프레임워크
- SPA(Single Page Application) 개발을 위한 프레임워크

- 가상 DOM(Virtual DOM) 개념에 의해 동작하는 프론트엔드 프레임워크
- 가상 DOM 개념은 Angular, Vue.js와 같은 프론트엔드 프레임워크에서도 볼 수 있음
- 가상 DOM을 이해하려면 리액트 프로제트의 동작 방식에 대한 이해가 필요

4. 컴포넌트

- 재사용이 가능한 블럭
- Header / Body / Footer 안에도 많은 컴포넌트들이 존재

- MPA 기반 애플리케이션은 하나의 변경사항을 위해 전체 페이지를 리로딩해야 하지만,
- SPA 기반 애플리케이션은 컴포넌트를 단위로 변경사항을 반영한다.

5. 리액트의 동작 방식

- React JS: 어플리케이션이 아주 interactive 하도록 만들어주는 library.
- ReactDOM: 모든 React element들을 HTML body에 둘 수 있도록 해준다.
- render: React element 를 가지고 HTML 로 만들어 배치한다. (사용자에게 보여준다)

- React JS는 우리가 해왔던 방식을 거꾸로 한다.
  - 바닐라 JS: HTML - JavaScript - HTML
  - React JS: JavaScript - HTML

- JavaScript 를 이용해 element 를 생성했고 React JS가 그걸 HTML로 번역!




1.2 JavaScript

1. 객체의 복사

- 객체는 값이 아닌 주소값을 그대로 복사한다.

```
const obj1 = {
    somekey1: 10,
    somekey2: '안녕',
};

const obj2 = obj1;
obj1.somekey1 += 1;
console.log('obj1', obj1);
console.log('obj2', obj2);
```

- 따라서 원본을 유지하고 복사하기 위해서는 얕은 복사 또는 깊은 복사를 해주어야 한다.

```
const obj3 = { ...obj1 }
const obj4 = JSON.parse(JSON.stringify(obj1));

obj1.somekey1 += 1;

console.log('obj3', obj3);
console.log('obj4', obj4);
```

2. 템플릿 리터럴

- 백틱을 사용하면 문자열 안에 자바스크립트의 변수를 사용 가능
- 멀티라인을 쓸 때도 유용

```
let name = '최인규';
console.log(`안녕하세요 제 이름은 ${name} 입니다.`);

let someValue = `
안녕하세요~
제 이름은 ${name}입니다.`;
```

3. 객체, 배열 비구조화 (Object/Array Destructuring)

- 객체에서 구조 분해 할당을 하려면 key 값이 같아야 한다.

```
const person = {
    name: '최인규',
    age: 17,
};

const { name, age } = person;
```

- 배열에서 구조 분해 할당을 하려면 위치가 같아야 한다.

```
const arr = [1, 2, 3, 4, 5];
const [하나, 둘, 셋] = arr;

console.log('하나', 하나);
console.log('둘', 둘);
console.log('셋', 셋);
```

4. 전개 연산자 Spread Operator

```
let [name, ...info] = ['최인규', 17, '서울'];


let names = ['홍길동', '김철수', '이영희'];
let students = ['최인규', ...names, ...names];
console.log(students);

let arr = ['사과', '오렌지', '딸기', '수박', '메론'];
let [사과, 오렌지, ...rest] = arr;
console.log(rest);


let inkyu = {
    name: '최인규',
    age: 17,
    region: '서울',
    email: '484342@gmail.com'
};
const { name, age, ...rest } = inkyu;
console.log(rest)
```

5. 화살표 함수 Arrow Functions

```
const mysum1 = (x, y) => x + y;
console.log(((x, y) => x + y)(1, 2));
console.log(mysum1(1, 2));

const mysum2 = (x, y) => ({ x, y });
const mysum3 = (x, y) => ({ x: x, y: y });
const mysum4 = (x, y) => { return { x: x, y: y } };
const mysum5 = function (x, y) { return { x: x, y: y } };
function mysum6(x, y) { return { x: x, y: y } };

console.log(mysum2(1, 2));
console.log(mysum3(1, 2));
console.log(mysum4(1, 2));
console.log(mysum5(1, 2));
console.log(mysum6(1, 2));
```

6. 배열 메서드

map()

- 배열 안의 요소들을 처리하여 새 배열을 만들 때 사용 (재배열)
- 요소 하나하나 로직을 돌려 재배열

```
const arr = [1, 3, 5, 7, 9];
const mapArr = arr.map((x) => x + 1)
console.log(mapArr);
```

```
const arr = [1, 2, 3, 4, 7];
const isOddArr = arr.map((x) => x % 2 == 1)
console.log(isOddArr);
```

6. 배열 메서드

filter()

- 콜백의 조건에 충족하는 값만 뽑아온다.
- 일치하는 애들만 뽑음

```
const arr = [1, 2, 3, 4, 5, 6];
const oddArr = arr.filter((x) => x % 2 === 1);
console.log(oddArr); //[1, 3, 5]

const arr = ['최인규', '홍길동', '김철수', '이영희'];
const kimArr = arr.filter((x) => x[0] === '김');
console.log(kimArr);
```

6. 배열 메서드

reduce()

- 배열의 값을 줄여서 하나의 값으로 만든다.

```
const arr = [1, 2, 3, 4, 5];
const sum = arr.reduce((acc, cur) => acc + cur, 100);
console.log(sum);

const scores = [7, 10, 7, 8, 8];
const total = scores.reduce((a, b) => a + b);
const cnt = scores.length;
```

6. 배열 메서드

forEach()

- 배열의 각 요소에 대해 함수를 실행한다.

```
const arr = ['최인규', '김철수', '이영희'];

arr.forEach((x, i) => console.log(x.repeat(i + 1)));
```

1.3 패키지 매니저

1. npm? yarn?

- 둘 다 Javascript 패키지 관리도구
- yarn은 npm의 역할과 동일하지만 npm보다 성능적으로 개선된 패키지 매니저

```
npm install -g yarn

yarn add [패키지 이름]
```

- npm
  - node.js 설치시 자동으로 생성
  - node package manager의 약자
  - npm 플렛폼 자체
- yarn
  - 2016년 페이스북에서 개발한 패키지 관리자
  - npm과 호환성이 좋고, 속도 및 안전성 측면에서 npm보다 월등히 좋음

2. 명령어 비교

명령어	npm	yarn
dependencies 설치	npm install	yarn
패키지 설치	npm install [패키지명]	yarn add [패키지명]
dev 패키지 설치	npm install --save-dev [패키지명]	yarn add --dev[패키지명]
글로벌 패키지 설치	npm install --global [패키지명]	yarn global add [패키지명]
패키지 제거	npm uninstall [패키지명]	yarn remove [패키지명]
글로벌 패키지 제거	npm uninstall --save-dev [패키지명]	yarn global remove [패키지명]
업데이트	npm update	yarn upgrade
패키지 업데이트	npm update [패키지명]	yarn upgrade [패키지명]

1.4 React 시작하기

1. Create React App

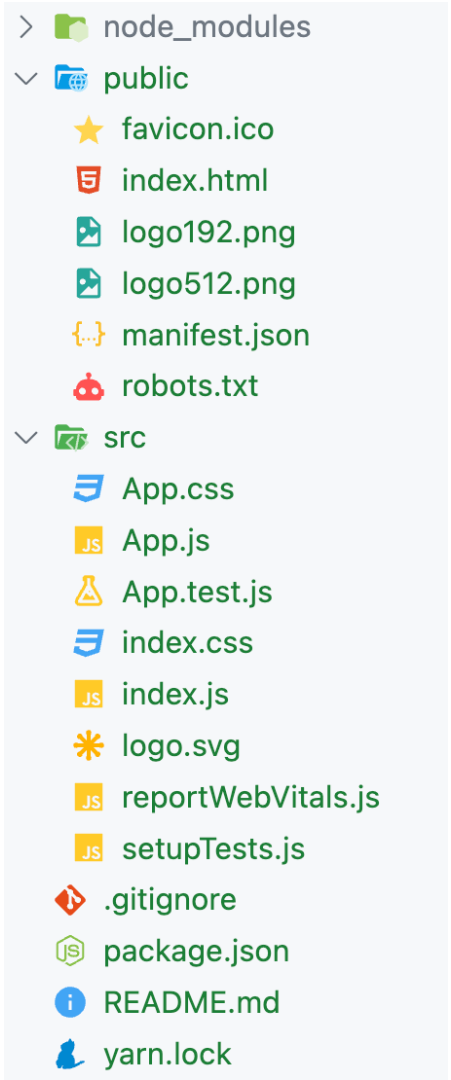
- CRA는 React 개발 시 사용되는 보일러플레이트(BoilerPlate) 방식
  - 보일러플레이트 : 프로젝트를 빠르게 시작하고, 일관된 구조와 설정을 가진 애플리케이션을 개발할 수 있도록 도와준다.

- 한 줄의 명령어로 React 프로젝트에 필요한 개발 필수 요소를 자동으로 구성
- 이전에 작성한 코드를 후속 모듈에 적용할 수 있는 재사용 가능한 프로그래밍 코드
- webPack, babel, eslin 등과 같은 도구들을 자동으로 설치

```
ls
mkdir 폴더이름
cd 폴더이름
yarn create react-app .

yarn start
```

2. 프로젝트 구조

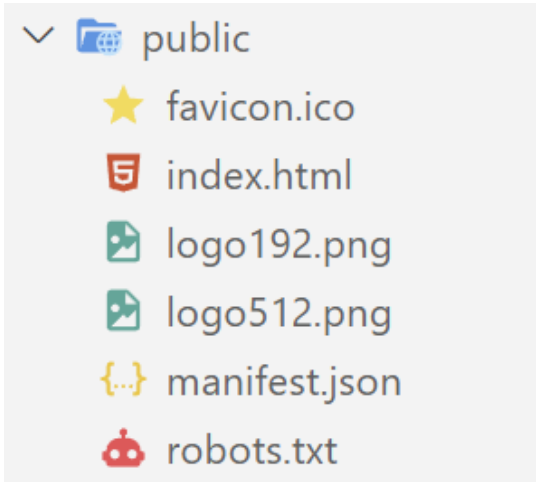


package.json에 있는 모듈들이 설치되는 폴더
정적 파일들이 들어가는 폴더

동적 파일들이 들어가는 폴더

github에 올릴 때 무시할 파일들을 설정하는 파일
프로젝트의 정보를 담고 있는 파일
프로젝트의 설명을 담고 있는 파일
npm 버전을 고정시키는 파일

2. 프로젝트 구조



웹 페이지 아이콘 파일

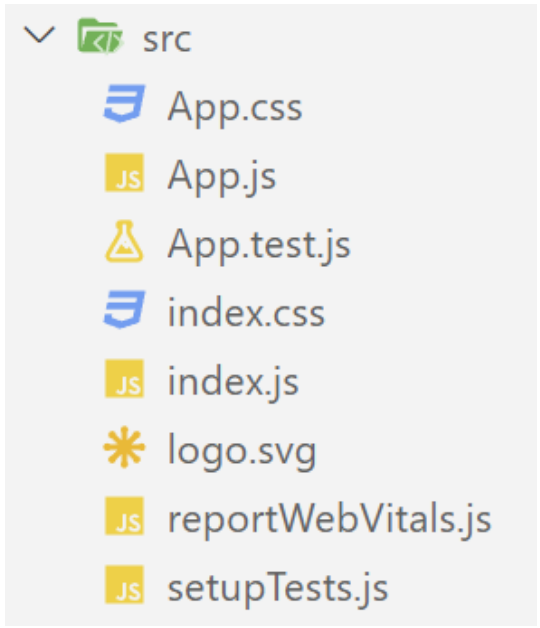
메인 페이지 파일(웹 페이지의 시작점)

리액트 로고 파일(192x192)

리액트의 로고 파일(512x512)

웹 페이지의 정보를 담은 파일

웹 크롤링 방지를 위한 설정 파일



리액트의 컴포넌트 css 파일

리액트의 컴포넌트 파일

리액트의 테스트 파일

리액트의 css 파일

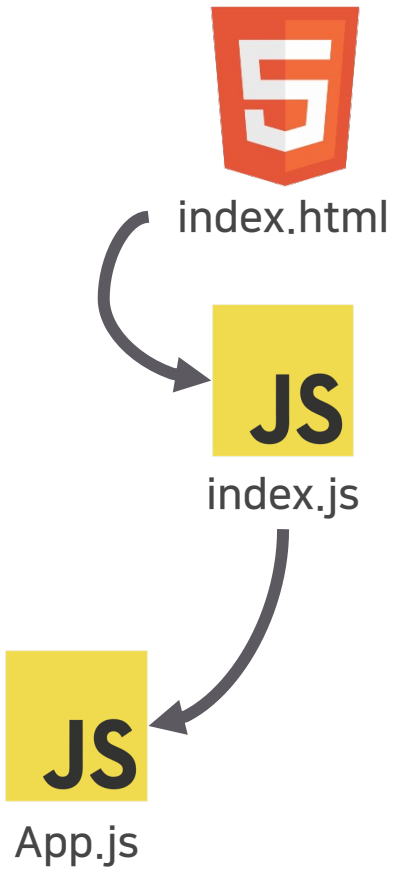
리액트의 메인 파일

로고 파일

성능 측정 파일

테스트 설정 파일

2. 프로젝트 구조




```
<div id="root"></div>
```

```
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

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

3. 기본 세팅

```
// App.js
import "App.css";
function App() {
    return (
        <div>
            <h1>ReactJS 시작</h1>
        </div>
    );
}
export default App;
```

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>
);
```

1.5 JSX 문법

1. JSX?

- JSX : React JS를 위해 확장된 문법
  - 컴포넌트 기반 개발
  - JavaScript 코드와 HTML을 결합해 사용

- JSX의 주요 특징
  - 중괄호 { }를 사용하면 JavaScript 표현식 삽입이 가능
  - 일반적인 HTML 태그와 속성 사용이 가능
  - 컴포넌트 렌더링 가능
  - 태그 속성을 통한 이벤트 핸들링

1. JSX?

- JSX를 사용하면 편리하게 화면을 렌더링할 수 있다.

```
function App() {
    return (
        <div>
            Hello <b>react</b>
        </div>
    );
}
```

```
function App() {
    return React.createElement("div", null, "Hello", React.createElement("b", null, "react"));
}
```

2. JSX 문법

- 리액트는 딱 하나의 HTML 파일만 존재 한다. (public 폴더 아래 index.html)
- JSX문법을 사용하여 React 요소를 만들고 DOM에 렌더링 시켜서 뷰를 그리게 된다.

- 단일태그에서도 태그를 닫자!

```
<input type='text' />
```

- 하나의 엘리먼트만을 반환하자!

```
function App() {
    return (
        <div className="App">
            <p>안녕하세요</p>
            <input type='text' />
        </div>
    );
}
```

2. JSX 문법

- JSX는 HTML보다는 JavaScript에 가깝다.
- class=" "이 아닌 className=" "으로, for=" " 대신 htmlFor=" "로 작성해야 한다.

```
<div calssName="title">안녕하세요</div>
```

- 인라인 스타일은 객체 형태로 작성하며, -을 사용할 수 없다.

```
<div className="someClass" style={{
            color: 'red',
            backgroundColor: 'tomato'
        }}>안녕하세요</div>

const App = () => {
    const style = {color: 'red', backgroundColor: 'tomato'}
    return (
        <div style={style}>안녕하세요</div>
    );
}
```

2. JSX 문법

- JSX 내부에서 중괄호를 열어 자바스크립트를 사용할 수 있다.

```
function App() {
    const myName = '최인규';
    const num = 1'
    return (
        <div>
            <p>hello {myName}!</p>
            <p>{number > 10 ? number + '은 10보다 크다' : number + '은 10보다 작다'}</p>
        </div>
    );
}
```

2. JSX 문법

- if문 대신에 삼항 조건 연산자를 이용한다.

```
function App() {
    const name = '리액트';
    return (
        <div>
            {name === '리액트' ? (
                <h1>리액트입니다.</h1>
            ) : (
                <h2>리액트가 아닙니다.</h2>
            )}
        </div>
    );
}
export default App;
```

2. JSX 문법

- JSX 내부에서 중괄호를 열어 자바스크립트를 사용할 수 있다.

```
function App() {
    const name = '리액트';
    return <div>{name === '리액트' ? <h1>리액트입니다.</h1> : null}</div>;
}
export default App;
```

```
function App() {
    const name = '리액트';
    return <div>{name === '리액트' && <h1>리액트입니다.</h1>}</div>;
}
export default App;
```

2. JSX 문법

- 함수에서 undefined만 반환하여 렌더링하는 상황을 만들면 안된다.

```
import './App.css';
function App() {
    const name = undefined;
    return name;
}
export default App;
```

```
import './App.css';
function App() {
    const name = undefined;
    return name || '값이 undefined입니다.';
}
export default App
```

2. JSX 문법

- 주석은 {/* ... */} 형식으로 작성

```
import './App.css';
function App() {
    const name = '리액트';
    return (
        <>
            {/* 주석은 이렇게 작성합니다. */}
            <div className="react" // 시작 태그를 여러 줄로 작성하게 된다면 주석을 작성할 수 있습니다.
            >{name}</div>
        // 하지만 이런 주석이나
        /* 이런 주석은 페이지에 그대로 나타나게 됩니다. */
        </>
    );
}
```

3. import

- import를 이용해 직접 작성한 파일이나 자원을 로드할 수 있다.

```
import ... from "./..." -> 동일 경로
import ... from "../..." -> 상위 경로
import { Fragment } from 'react' -> node_modules / react / index.js
```

1.6 Components

1. 컴포넌트

- 컴포넌트는 리액트의 핵심!
- UI 요소를 표현하는 최소한의 단위이며 화면의 특정 부분이 어떻게 생길지 정하는 선언체
- 개념적으로 컴포넌트는 JavaScript 함수와 유사 (쉽게 말해 리액트는 html을 return 하는 함수)

- 컴포넌트를 만들 때 반드시 가장 첫 글자는 대문자! (파스칼케이스)
- 폴더는 소문자로 시작하는 카멜케이스로 작성!
- 컴포넌트 파일은 .js 확장자가 아닌 .jsx 확장자를 쓰는 것이 보기에 좋다.

1. 컴포넌트

- 클래스형 컴포넌트와 함수형 컴포넌트

```
import './App.css';
function App() {
    const name = '리액트';
    return <div className="react">{name}</div>
}
export default App;
```

```
import { Component } from 'react';
class App extends Component {
    render() {
        const name = 'react';
        return <div className="react">{name}</div>;
    }
}
export default App;
```

2. 첫 컴포넌트 생성하기

- 컴포넌트를 만들고, App.js에서 import해서 사용

```
const MyComponent = () => {
    return <div>나의 새롭고 멋진 컴포넌트</div>;
};
export default MyComponent;
```

```
import MyComponent from './MyComponent';
const App = () => {
    return <MyComponent />;
};
export default App;
```

1.7 props

1. props

- 부모 컴포넌트가 자식 컴포넌트에게 물려준 데이터
- 컴포넌트 간의 정보 교환 방식으로 데이터를 교환하는 한 가지 방법

- props는 반드시 위에서 아래 방향 즉, [부모] ➜ [자식] 방향 (단반향) 으로 흐른다.
- props는 반드시 읽기 전용으로 취급하며, 변경하지 않아야 한다.

1. props

```
import MyComponent from './MyComponent';
const App = () => {
    return <MyComponent name="React" />;
};

export default App;
```

```
const MyComponent = props => {
    return <div>안녕하세요. 제 이름은 {props.name}입니다.</div>;
};

export default MyComponent;
```

2. defaultProps

- 기본값 설정

```
const MyComponent = props => {
    return <div>안녕하세요. 제 이름은 {props.name}입니다.</div>;
};

MyComponent.defaultProps = {
    name: '기본 이름'
};

export default MyComponent;
```

3. props children

- 태그 사이의 내용을 보여 주는 children
- 비구조화 할당 문법을 통해 props 내부 값 추출 가능

```
import MyComponent from './MyComponent';
const App = () => {
    return <MyComponent>리액트</MyComponent>;
};

export default App;
```

```
const MyComponent = ({ name, children }) => {
    return (
        <div>
            안녕하세요. 제 이름은 {name}입니다. <br />
            children 값은 {children}입니다.
        </div>
    );
};
MyComponent.defaultProps = {
    name: '기본 이름'
};
export default MyComponent
```

4. props children의 용도

- props children은 Layout 컴포넌트를 만들 때 자주 사용한다.

```
const App = () => {
    return (
        <Layout>
            <div>안녕하세요</div>
        </Layout>
    );
};

const Layout = (props) => {
    const style = {
        backgroundCololr: 'tomato',
        padding: '1rem',
        border: '2px dashed red',
    };
    return (
        <div>
            <header style={style}> 항상 출력 되는 헤더 부분 </header>
            {props.children}
        </div>
    );
};

export default App;
```

5. propTypes

- propTypes를 통한 props 검증
- isRequired를 사용하여 필수 propTypes 설정

```
import PropTypes from 'prop-types';
const MyComponent = ({ name, favoriteNumber, children }) => {
    <div>
        안녕하세요. 제 이름은 {name}입니다. <br />
        children 값은 {children}입니다.
        <hr />
        제가 좋아하는 숫자는 {favoriteNumber}입니다.
    </div>
}
MyComponent.defaultProps = {
    name: '기본 이름'
};
MyComponent.propTypes = {
    name: PropTypes.string,
    favoriteNumber: PropTypes.number.isRequired
};
export default MyComponent;
```

```
import MyComponent from './MyComponent';
const App = () => {
    return <MyComponent name={3}>리액트</MyComponent>
};

export default App;
```

3. props drilling

- props는 [부모] → [자식] 컴포넌트간 데이터 전달이 이루어지는 방법인데,
- [부모] → [자식] → [그 자식] → [그 자식의 자식]에서 부모 데이터를 이용하기 위해서는 무려 3번이나 데이터를 내려줘야 한다. 이것을 `props drilling` 이라고 한다.

```
import React from 'react';

function App() {
    const name = '최인규';
    return <Inkyu />;
};

const Inkyu = (props) => {
    return <Child parentName={name} />;
};

const Child = (props) => {
    const master = props.parentName;
    return <Puppy mastersName={master} />;
};

const Puppy = (props) => {
    return <div>나는 {props.parentName}의 강아지 입니다.</div>;
};

export default App;
```

4. props drilling의 문제점

- 부모의 `props`를 전달 받는 과정을 보면, 불필요한 중간다리 역할들이 존재한다.
- 이렇게 깊이가 너무 깊어지면 이 prop이 어떤 컴포넌트로부터 왔는지 파악이 어렵다.
- 어떤 컴포넌트에서 오류가 발생할 경우 추적이 힘들어지니 대처가 늦게 된다.

- 그래서 등장한 것이 react context API이다.
- 이를 useContext hook을 이용하면 쉽게 `전역 데이터를 관리`할 수 있다.

1.8 state

1. state

- 앱 또는 컴포넌트의 ‘상태’
- 데이터가 가진 값, 모달이 열려 있는지 여부, 어떤 값이 선택되었는지 등... 모든 것들이 '상태'이다.
- 그리고 모든 ‘상태’에 따라 특정 값을 ‘render’한다.

1. state

- state는 컴포넌트 내부에서 바뀔 수 있는 값으로, 렌더링을 일으키는 동적인 값(변수)!
- state는 컴포넌트 내부에서 선언되어, 컴포넌트의 렌더링 결과에 영향을 준다.

```
import React, { useState } from 'react';
const numberState = useState(0);
```

- useState 를 이용하면 두 개의 원소로 구성된 배열이 반환된다.
  - numberState[0] : 현재 상태값 (기본값)
  - numberState[1] : 상태값을 변경할 수 있는 Setter 함수

- 비구조화 할당을 통해 아래와 같이 작성하는 것이 일반적

```
import React, { useState } from "react";
const [state, setState] = useState(initState);
```

2. state 사용하기

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

const App = () => {
    const [count, setCount] = useState(0);
    return (
        <div className="App">
            {count}
            <button onClick={() => setCount(count + 1)}>Click Me!</button>
        </div>
    );
};

export default App;
```

2. state 사용하기

```jsx
import Say from './Say';

const App = () => {
    return <Say />;
};

export default App;
```

```jsx
import { useState } from 'react';
const Say = () => {
    const [message, setMessage] = useState('');
    const onClickEnter = () => setMessage('안녕하세요');
    const onClickLeave = () => setMessage('안녕히 가세요');
    return (
        <div>
            <button onClick={onClickEnter}>입장</button>
            <button onClick={onClickLeave}>퇴장</button>
            <h1>{message}</h1>
        </div>
    );
};
export default Say;
```

3. state 활용 실습

- 아이디와 비밀번호를 입력하고 [로그인] 버튼을 누르면 alert로 고객이 입력한 값을 알려주기
  - alert 내용 : "고객님이 입력하신하신 아이디는 '~~~' 이며, 비밀번호는 '~~~' 입니다."

- 아이디와 비밀번호 필드의 값은 state로 관리되어야 하며, 변경이 일어날 때마다 setState가 진행된다.
- alert를 띄운 후에는, 아이디와 비밀번호 영역을 초기화되어야 한다.

4. state를 사용할 때 주의사항

```
// 객체 다루기
const object = { a: 1, b: 2, c: 3 };
const newObject = { ...object, b: 2 }; // 사본을 만들어서 값만 덮어 쓰기

// 배열 다루기
const array = [
    { id: 1, value: true },
    { id: 2, value: true },
    { id: 3, value: false }
];

// 새 항목 추가
let nextArray = array.concat({ id: 4 });

// id가 2인 항목 제거
nextArray.filter(item => item.id !== 2);

// id가 1인 항목의 value를 false로 설정
nextArray.map(item => (item.id === 1 ? { ...item, value: false } : item));
```