React

React란?

React는 Facebook에서 개발한 **JavaScript 라이브러리**로, 웹 애플리케이션의 UI(User Interface)를 효율적으로 만들기 위해 사용된다.

React를 사용하면 웹 페이지의 복잡한 구조를 관리하기 쉽게 만들어 준다. 웹뿐 아니라 모바일 앱 개발(React Native)에도 사용된다.

React는 HTML과 같은 구조를 JavaScript 코드로 작성하게 하고, 화면을 업데이트하는 과정을 자동화해 주어 **개발자의 생산성을 크게 향상**시킬 수 있다.

React의 주요 개념과 특징

1. 상태(State)

React의 상태(State)는 **컴포넌트 내부에서 관리되는 데이터**로, UI의 동적인 변화를 반영하는 데 사용된다. 상태는 **컴포넌트의 현재 상태를 나타내는 값**이며, React에서 **상태가 변경되면 해당 컴포넌트와 그 자식 컴포넌트가 다시 렌더링** 된다.

상태의 주요 특징

1. **컴포넌트 내부에서 관리**:
   - 상태는 컴포넌트 내부에서 선언되고, 해당 컴포넌트에만 영향을 미친다.
   - 다른 컴포넌트와 데이터를 공유하려면 `props`를 통해 전달해야 한다.
2. **변경 가능 (mutable)**:
   - React의 상태는 변경될 수 있지만, 직접 값을 수정하지 않고 **상태를 변경하는 함수(setter)**를 사용해야 한다.
3. **UI 업데이트**:
   - 상태가 변경되면 React는 해당 컴포넌트와 관련된 UI를 **자동으로 업데이트한다.**

상태 변경의 규칙

1. **직접 수정 불가**

```
const [count, setCount] = useState(0);
count = count + 1; // ❌ 상태 직접 수정은 React가 감지하지 못함.
setCount(count + 1); // ✅ setter 함수를 사용해 수정.
```

2. **비동기적으로 동작**:
   - `setState`나 `setCount` 호출은 비동기적으로 처리되므로, 바로 직후에 상태가 변경된 값을 확인 할 수 없다.

```
...
setCount(count + 1); // 상태 업데이트 요청
console.log(count); // 아직 업데이트되지 않은 이전 값 출력
```

2. 컴포넌트(Component)

컴포넌트란 화면의 한 부분을 담당하는 독립적인 코드 블록을 의미한다.

컴포넌트를 재사용 가능하도록 만들어 코드를 효율적이고 유지보수하기 쉽게 만들어준다.

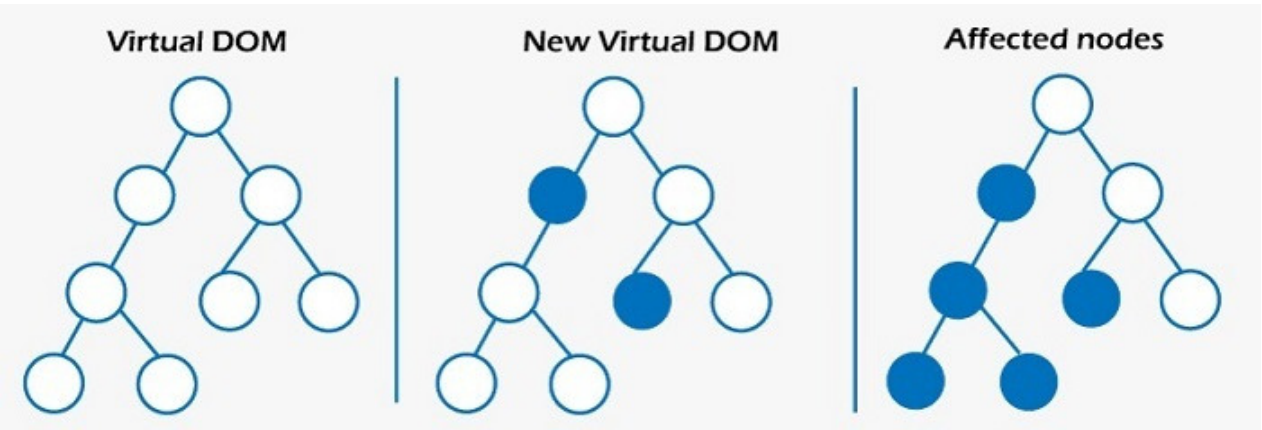
3. 가상 DOM (Virtual DOM)

DOM(Document Object Model) : 브라우저가 화면의 구조를 이해하는 방식 (문서의 구조화된 표현)

React는 **Virtual DOM**이라는 경량화된 "가상" 버전의 DOM을 사용한다.

React는 화면이 변경될 때 실제 DOM을 바로 업데이트하지 않고, **실제 DOM을 복사해둔 가벼운 가상 DOM**에서 **변경된 부분을 계산한 뒤 최소한의 변경만 실제 DOM에 적용**한다.

결과적으로 **더 빠르고 효율적으로** UI를 업데이트할 수 있다.



4. 단방향 데이터 흐름

React에서는 데이터가
부모에서 자식으로만 흐른다는 특징이있다. 이를 **단방향 데이터 흐름**이라고 부른다.

부모 컴포넌트는 **props를 통해 데이터를 자식에게 전달하고, 자식** 컴포넌트는 **props를 받아서 UI를 생성하거나 처리**한다.

```
// 자식 Greeting 컴포넌트
function Greeting(props) {
    return <h1>안녕하세요, {props.name}님!</h1>;
}

// 부모 App 컴포넌트
function App() {
    return (
        <div>
            <Greeting name="홍길동" /> - 부모쪽에서 자식인 Greeting 컴포넌트를 호출함
            <Greeting name="이순신" /> - 호출 시 name="홍길동", name="이순신"과 같은 데이터(props) 전달함
        </div>
    );
}
```

5. JSX (JavaScript XML)

JSX는 JavaScript에서 HTML을 작성할 수 있는 특별한 문법이다. React 컴포넌트를 만들 때, 화면에 보여질 구조를 JavaScript 코드로 작성할 수 있다.
JavaScript의 기능과 HTML을 결합할 수 있어
더 직관적이고 생산적인 코드 작성이 가능하다.

```
// 일반 HTML
<h1>안녕하세요!</h1>

// JSX로 작성
const element = <h1>안녕하세요!</h1>;
```

6. React Hooks

React Hooks는 **함수형 컴포넌트**에서 **상태(state)**와 **React의 생명주기(lifecycle)** 기능을 사용할 수 있도록 만든 API이다.

이전에는 클래스형 컴포넌트에서만 상태와 생명주기 기능을 사용할 수 있었지만, Hooks 덕분에 함수형 컴포넌트에서도 이를 간편하게 사용할 수 있게 되었다.

(useState, useEffect, useContext.. 등)

7. SPA (Single Page Application)

하나의 HTML 페이지로 동작하는 웹 애플레케이션을 SPA 라고 한다.

- **기존 방식(Multi-Page Application)**:
  - 사용자가 새로운 페이지로 이동할 때마다 **서버에 요청**하고, 서버가 **새로운 HTML 문서를 반환**받는다.
  - 페이지가 전환될 때마다 **전체 화면이 다시 로드**되므로, 페이지 로딩 시간이 발생하고 화면이 깜빡이는 현상이 나타난다.

- **React 방식(SPA)**:
  - React로 구축된 SPA는 처음 로드 시 **하나의 HTML 파일**과 필요한 JavaScript 파일을 서버로부터 받아온다.
  - 이후 사용자가 다른 페이지로 이동하면, 브라우저는 **페이지 전체를 새로 로드하지 않고** 필요한 데이터만 서버에서 가져온다.
  - React는 변경된 데이터만 DOM에 업데이트하고, 브라우저는 이 데이터를 사용해 **즉각적인 페이지 전환**을 제공한다.
  - 결과적으로 SPA는 더 빠르고 부드러운 사용자 경험을 제공한다.

React 기본 디렉토리 구조

my-react-app/
├── node_modules/ # 설치된 라이브러리 및 종속성
├── public/ # 정적 파일 (HTML, 이미지 등)
│ └── index.html # React 앱의 템플릿 파일
├── src/ # React 소스 코드
│ ├── App.css # App 컴포넌트의 스타일링 파일
│ ├── App.js # 메인 컴포넌트 (root-level component)
│ ├── index.css # 전역 스타일링 파일
│ ├── index.js # React 앱 진입점 (엔트리 파일)
│ └── logo.svg # 기본 로고 파일 (React 로고)
├── .gitignore # Git에서 무시할 파일/폴더 목록
├── package.json # 프로젝트 메타데이터 및 종속성 관리
├──
README.md # 프로젝트 설명 파일
└── yarn.lock 또는 package-lock.json # 패키지 버전 고정 파일

1. node_modules

- React 애플리케이션에 필요한 **모든 라이브러리**와 **종속성**이 저장되는 디렉토리.
- React, React DOM, Webpack, Babel 등의 라이브러리가 포함되어 있음
- **주의**: 직접 수정하지 하지 말 것

2. public

- 정적 파일을 저장하는 폴더로, React 앱이 로드될 때 사용됨
- 이 디렉토리에 있는 파일은 브라우저에서 그대로 제공됨.
- **주요 파일**:
  - `index.html`:
    - **React 애플리케이션이 처음으로 로드되는 HTML 파일.**
    - React의 모든 컴포넌트는 이 파일의 `<div id="root"></div>`에 렌더링됨.
  - **기타 정적 파일**:
    - 로고, 이미지, 폰트 등 정적 리소스를 배치할 수 있음.

3. src

- React 애플리케이션의 **주요 소스 코드**를 저장하는 폴더.
- 컴포넌트, 스타일, 로직 등 모든 작업이 이곳에서 이루어진다.
- **주요 파일**:

- `App.js`:
  - React 애플리케이션의 **최상위 컴포넌트.**
  - 모든 컴포넌트는 이곳에서 조합되고 렌더링된다.
- `App.css`:
  - `App.js`에 적용되는 기본 스타일 파일.
  - 초기 설정에서 React 로고와 기본 텍스트 스타일이 정의되어 있음.
- `index.js`:
  - React 애플리케이션의 **진입점 파일**.
  - React DOM 라이브러리를 사용해 `index.html`의 `<div id="root">`에 앱을 렌더링한다.
  - 앱의 루트 컴포넌트인 `<App />`를 호출합니다.
  - `React.StrictMode` 설정도 이곳에서 이루어진다.

> 📝 **React.StrictMode란?**
>
> `React.StrictMode`는 React에서 제공하는 내장 컴포넌트로, 애플리케이션에서 **잠재적인 문제를 식별**하고 **더 나은 코드를 작성하도록 돕는 개발 도구**.
>
> React의 **개발 환경에서만 활성화**되며, 프로덕션(배포 환경)에서는 영향을 미치지 않는다.

- `index.css`:
  - 애플리케이션의 **전역 스타일**을 정의하는 파일.
- `logo.svg`:
  - React 앱 초기 화면에서 보여주는 기본 로고 파일.

4. .gitignore

- Git에서 추적하지 않아야 할 파일 및 디렉토리를 정의.
- 보통 `node_modules/`, 로그 파일, 환경 설정 파일 등이 포함됨.

> 📝 React 앱을 생성하면 자동으로 **Git이 초기화된다!**
>
> 이유는 `create-react-app` 도구가 프로젝트를 설정할 때, Git을 기본적으로 초기화하도록 설계되었기 때문.

5. package.json

- 프로젝트의 **메타데이터**와 **종속성**을 관리하는 파일.
- 프로젝트 이름, 버전, 스크립트 정의 (`start`, `build`, `test` 등).
- 설치된 npm 패키지 및 버전 정보.

6. README.md

- 프로젝트 설명을 작성하는 파일.
- 프로젝트 사용 방법, 설치 방법, 주요 기능 등을 설명하는 데 사용된다.

7. yarn.lock 또는 package-lock.json

- 프로젝트에서 사용하는 라이브러리의 **의존성 트리**와 **버전 고정**을 관리하는 파일.
- 팀 내에서 동일한 환경을 유지하기 위해 자동 생성됨.