

React

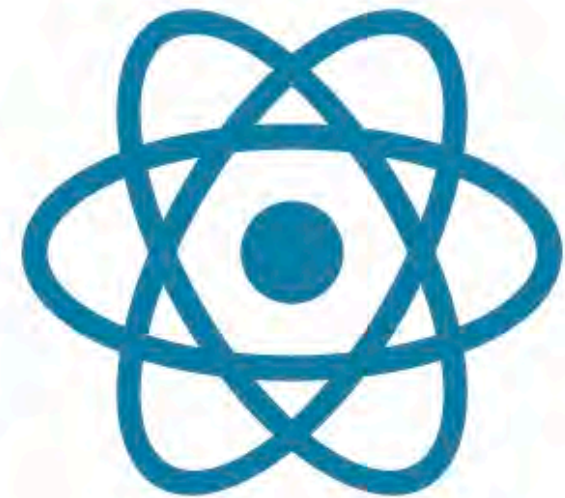
The library for web and native user interfaces

김 순곤

soongon@hucloud.co.kr

Introduction SPA & Components

SPA 와 컴포넌트 모델 소개



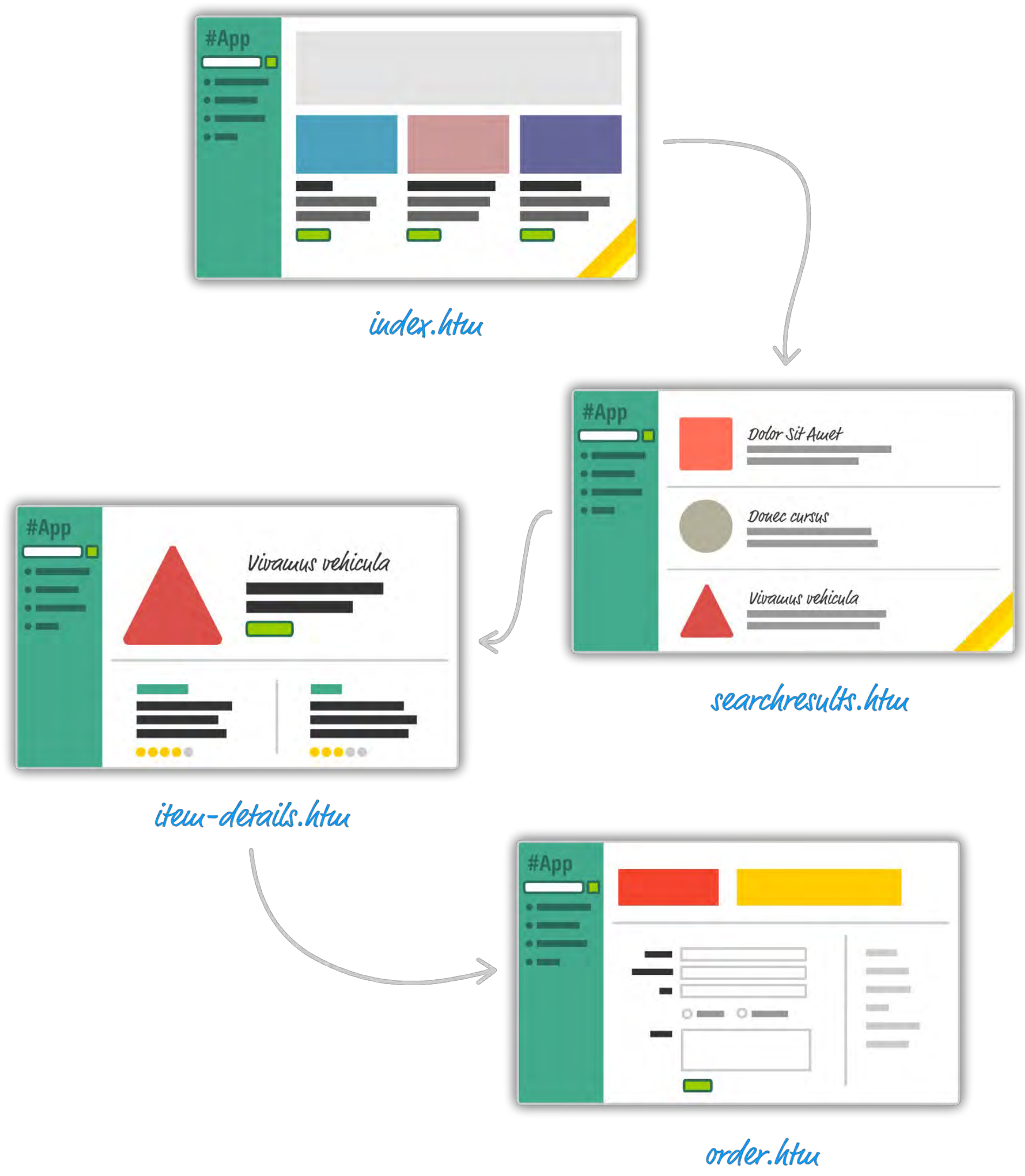
React

The library for web and native user interfaces

전형적인 웹앱 - 멀티 페이지



옛날 방식의 멀티 페이지 디자인

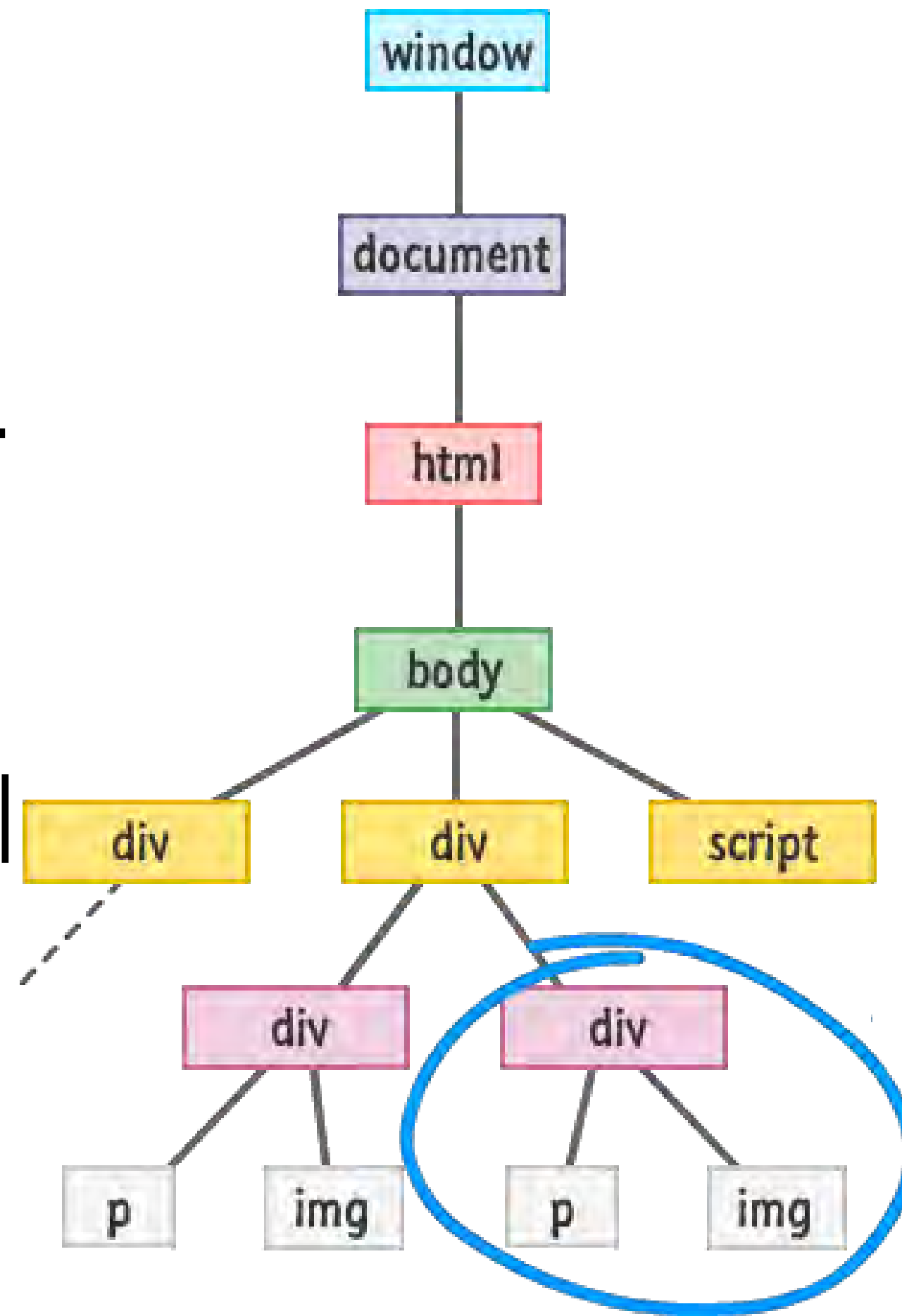


SPA - 싱글 페이지 앱



SPA 어플리케이션 개발 시 특징

- ☑ 데이터와 UI 동기화
- ☑ DOM 조작 - **DOM** 은 느리다
- ☑ HTML 에 데이터 바인딩하기
 - Interpolation



모던 웹 아키텍처 지원 프론트엔드 프레임워크의 특징

SPA

컴포넌트 기반 모델

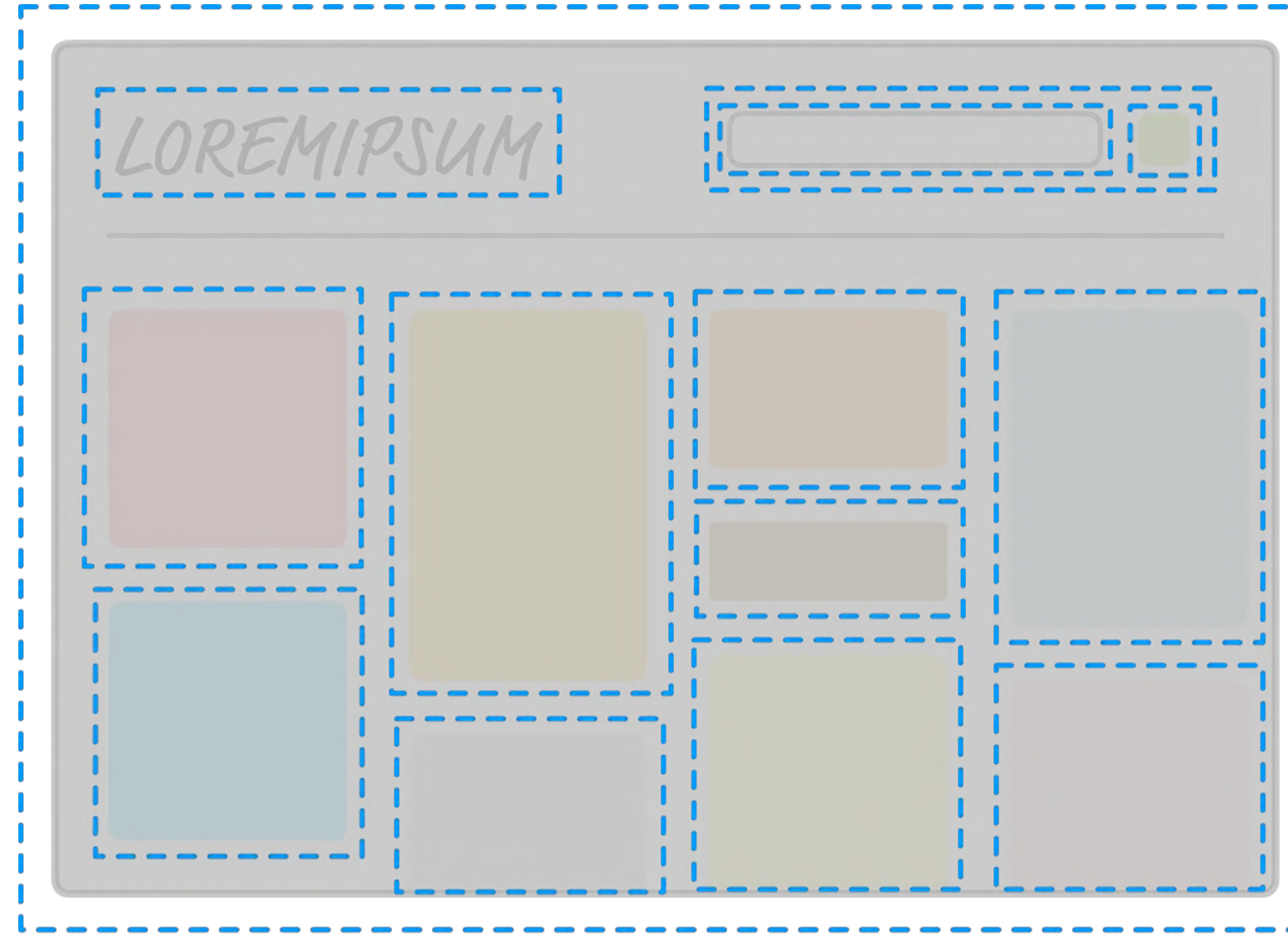
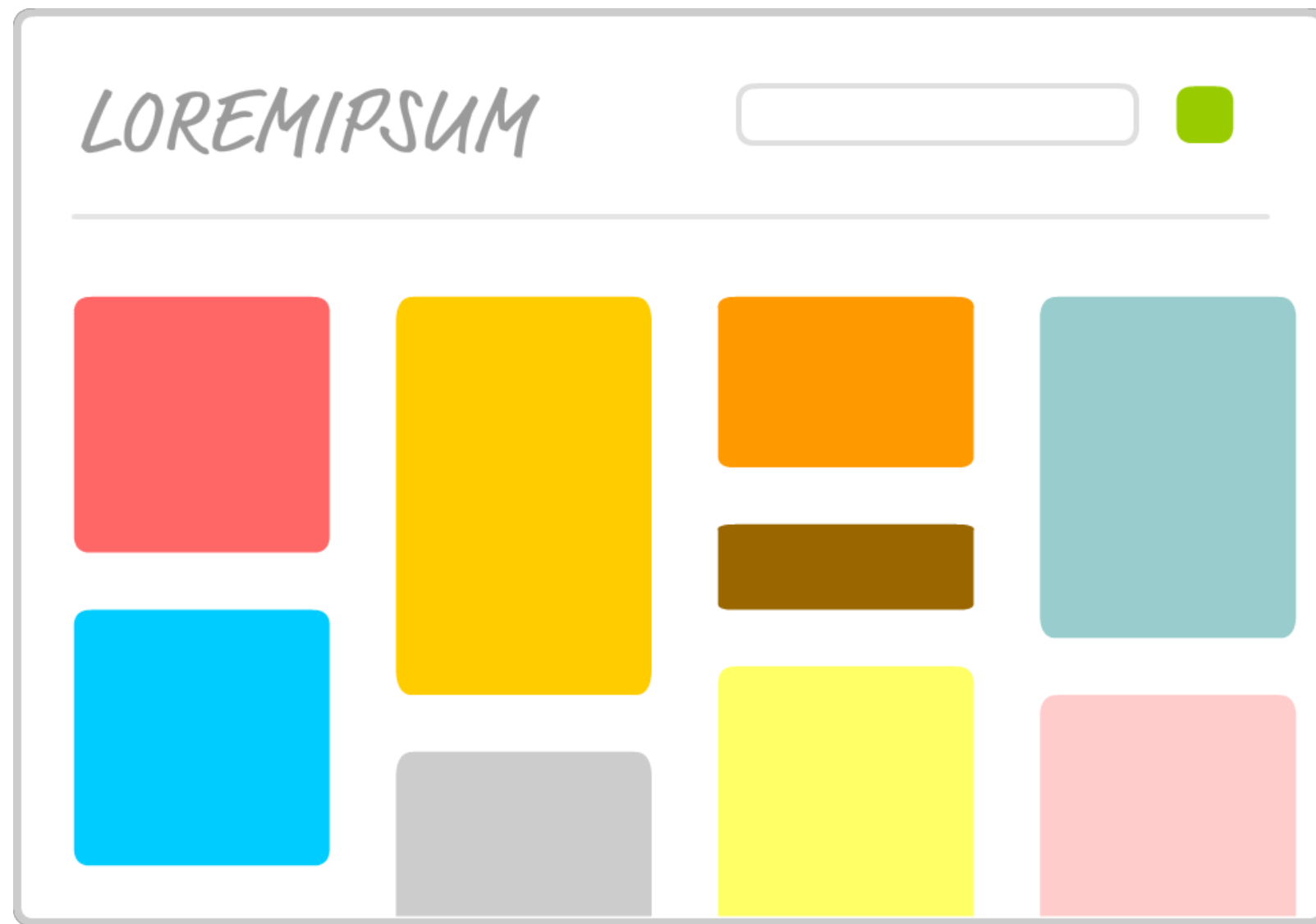
- ☑ 컴포넌트(화면)는 데이터와 레이아웃으로 구성되어 있다.
 - 데이터를 state 라고 한다.
 - state 는 화면의 요소(태그)에 바인딩 되어 있다.
 - state 값이 바뀌면 바인딩된 태그(화면)도 자동으로 다시 렌더링 된다.

컴포넌트 기반 프론트엔드 개발

작고 단단한 컴포넌트를 만드는 것

**이렇게 만들어진 컴포넌트간의
관계를 정의하고 유기적으로 연결하는 것**

UI 단위 - 컴포넌트

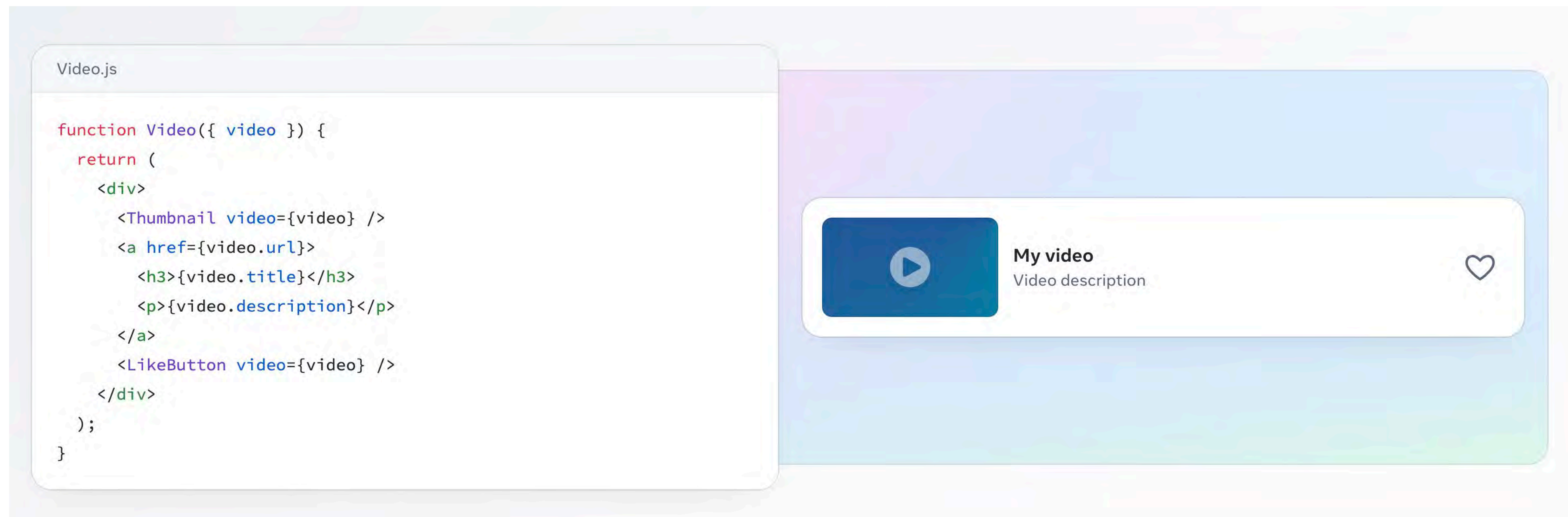


That's a lot of COMPONENTS!

REACT 특징

리액트의 특징

- ☑ 컴포넌트를 이용해 화면을 만드는 기술




리액트의 특징


- ☑ JSX(HTML like) 와 코드(JS, TS)를 이용해 컴포넌트를 작성

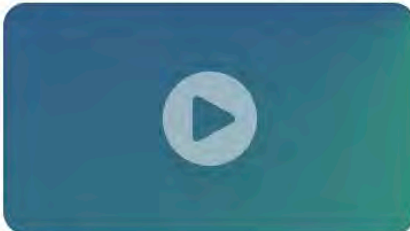
VideoList.js




```
function VideoList({ videos, emptyHeading }) {  
  const count = videos.length;  
  let heading = emptyHeading;  
  if (count > 0) {  
    const noun = count > 1 ? 'Videos' : 'Video';  
    heading = count + ' ' + noun;  
  }  
  return (  
    <section>  
      <h2>{heading}</h2>  
      {videos.map(video =>  
        <Video key={video.id} video={video} />  
      )}  
    </section>  
  );  
}
```

3 Videos

 **First video**
Video description

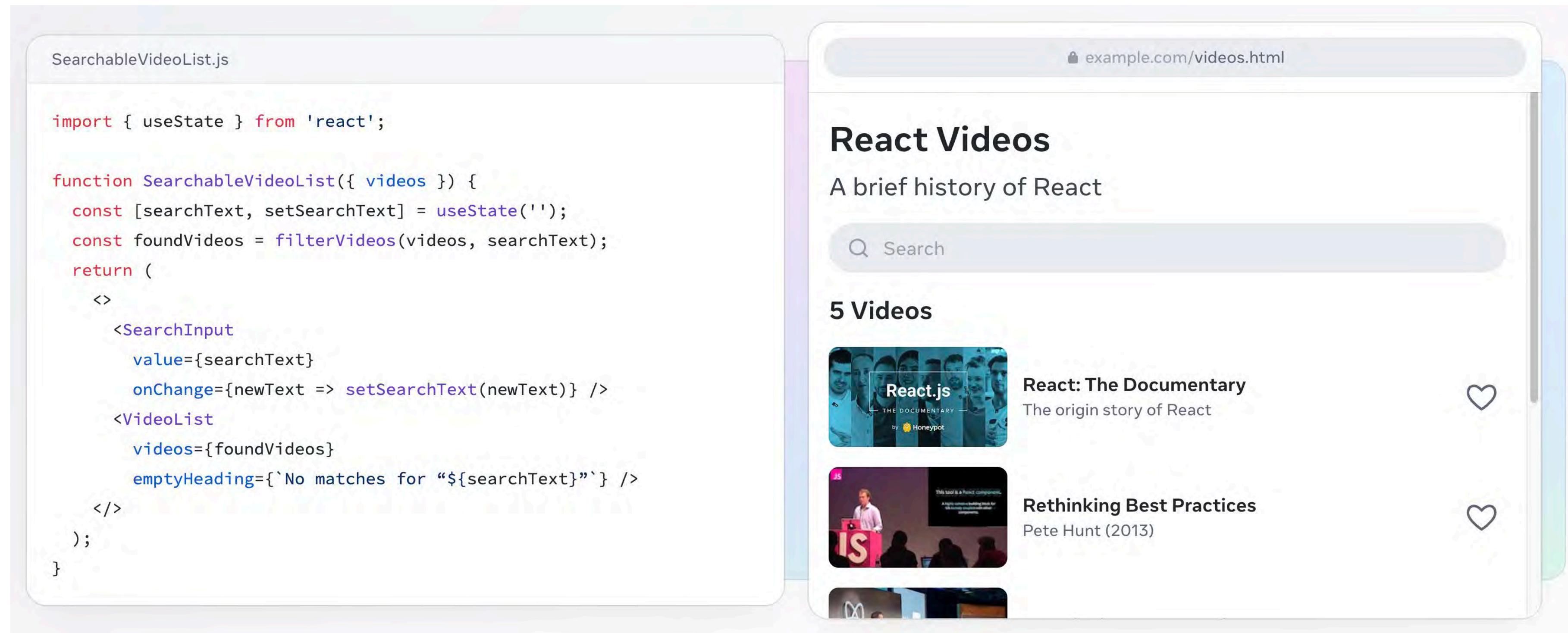
 **Second video**
Video description

 **Third video**
Video description

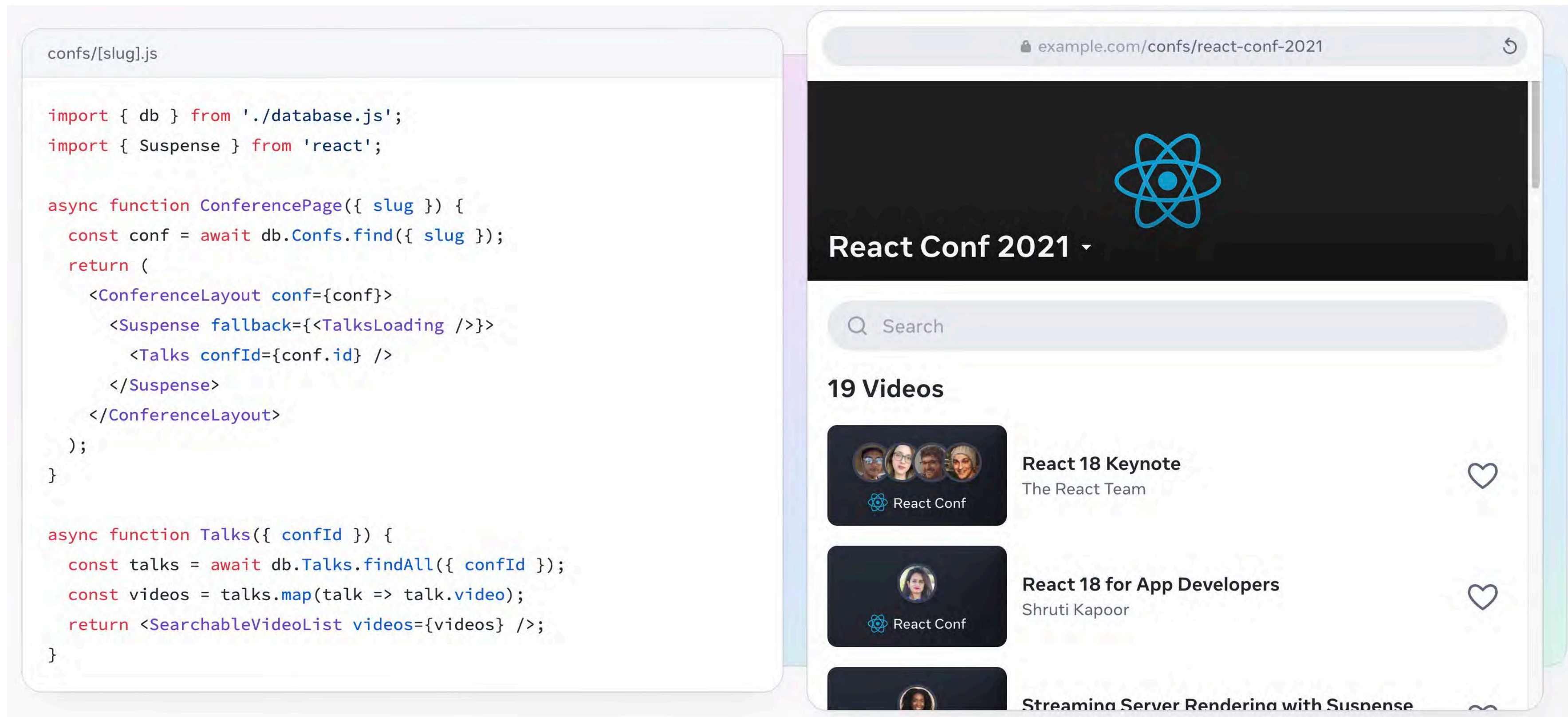
리액트의 특징

- ☑ 변경에 즉각 대응하여 화면이 변한다.



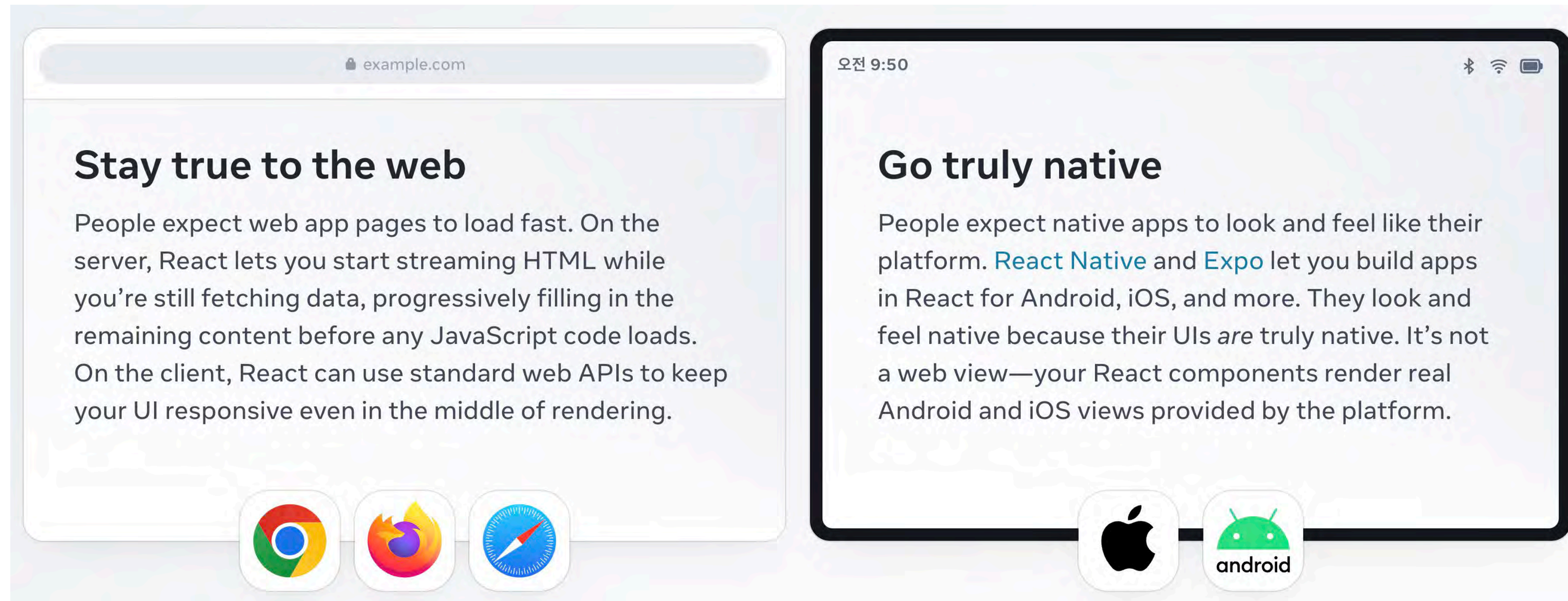
리액트의 특징

- ☑ 프레임워크를 이용해 풀스택 개발이 가능



리액트의 특징

☑ 다양한 플랫폼을 지원



React

Vue

Angular

Svelte

프론트엔드 프레임워크 비교

라이브러리 vs. 프레임워크

☑ 라이브러리 (react)

- 개발자가 사용
- 자유도 높음, 규격화 낮아짐

☑ 프레임워크 (angular, vue)

- 프레임워크에 맞춰 개발
- 자유도 낮음, 규격화 높아짐 (가독성 높아짐, Readability)

☑ 라이브러리 vs. 프레임워크

- 애플리케이션 복잡도가 증가되면 프레임워크가 적합

Angular vs. React

☑ Angular (typescript 기본 개발)

- From Google
- MVVM 프레임워크
- AOT 렌더링, 라우터, CLI
- 기본적으로 TypeScript 채택
- 디펜던시 인젝션 기능
- 글로벌 상태관리 시스템 부재 -> Flux, NgRx 채택 필요

☑ React, Vue (ES6 기본 개발)

- 컴포넌트 기반 개발에 중점을 둠 -> 프레임워크 도입에 노력과 시간을 줄여줌

React vs. Vue

- ☑ 2013년 Facebook 에서 오픈 릴리즈됨
 - JSX 패턴 (Javascript 에서 HTML과 CSS 를 작성하는 방법)
 - 모듈 방식의 애플리케이션 구축
- ☑ Vue -> svelte
 - 모듈식 코딩의 핵심을 취함
 - JavaScript 단일 파일에서 HTML, CSS 요소를 포함 (SFC)
 - 모듈식 개발 방식의 단순화

Node.js 설치

VsCode 설치

개발환경 설정

노드 설치

☑ Node.js 설치 : <http://nodejs.org>

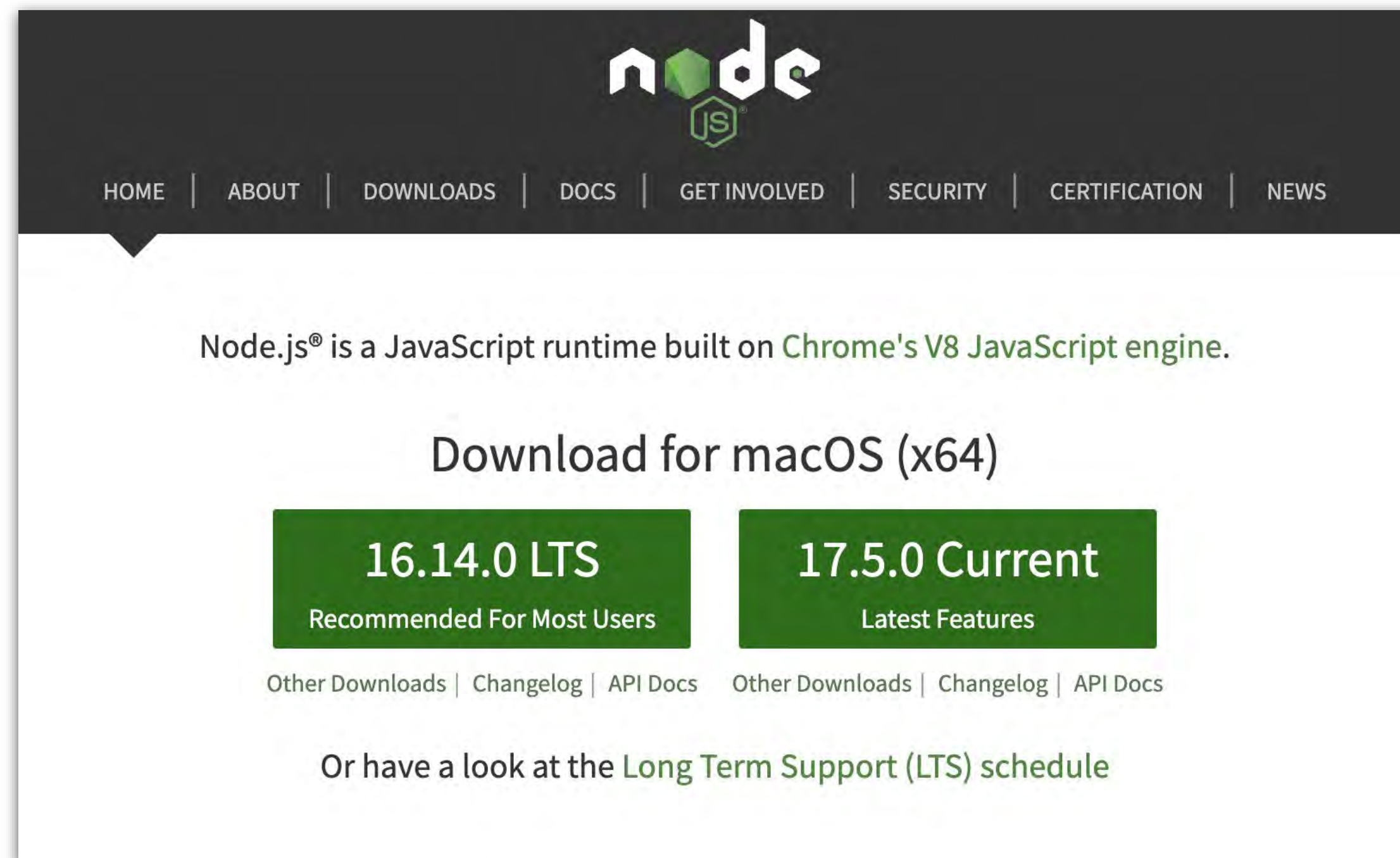
○ 16버전 이상 설치 권장

☑ 설치 확인


○ `node --version`

○ `npm --version`


위 두 가지
터미널에서 설치 확인




VS Code - <https://code.visualstudio.com/>

 Visual Studio Code

[Docs](#) [Updates](#) [Blog](#) [API](#) [Extensions](#) [FAQ](#) [Learn](#)

 Search Docs

 Download

Version 1.64 is now available! [Read about the new features and fixes from January.](#)

Code editing. Redefined.

Free. Built on open source. Runs everywhere.


[Download Mac Universal](#)
Stable Build


Other platforms and [Insiders Edition](#)


By using VS Code, you agree to its [license and privacy statement.](#)


EXTENSIONS: MARKETPLACE


@sort:installs


 **Python** 2019.6.24221 54.9M ★4.5
Linting, Debugging (multi-threaded...
Microsoft [Install](#)


 **GitLens — Git su...** 9.8.5 23.1M ★5
Supercharge the Git capabilities bui...
Eric Amodio [Install](#)


 **C/C++** 0.24.0 23M ★3.5
C/C++ IntelliSense, debugging, and...
Microsoft [Install](#)

 **ESLint** 1.9.0 21.9M ★4.5
Integrates ESLint JavaScript into V...
Dirk Baeumer [Install](#)

 **Debugger for C...** 4.11.6 20.6M ★4
Debug your JavaScript code in the ...
Microsoft [Install](#)

 **Language Sup...** 0.47.0 18.6M ★4.5
Java Linting, Intellisense, formatin...
Red Hat [Install](#)

 **vscode-icons** 8.8.0 17.2M ★5
Icons for Visual Studio Code
VSCode Icons Team [Install](#)

 **Vetur** 0.21.1 17M ★4.5
Vue tooling for VS Code
Pine Wu [Install](#)

blog-post.js — gatsby-graphql-app


src > components > JS blog-post.js > <function> > [e]blogPost


```
1 import { graphql } from 'gatsby'
2 import React from 'react'
3 import Image from 'gatsby-image'
4
5 export default ({ data }) => {
6   const blogPost = data.cms.blogPost
7   return (
8     <div>
9       {blogPost} {e} debug
10       blogPos {e} debugger
11       blogPos {e} decodeURI
12       <Image {e} decodeURIComponent
13     >
14     <h1>{blog {e} defaultStatus
15     <div>Post {e} delete
16     <div> dang {e} departFocus
17     </div>
18   )
19 }
20
21 export const query = graphql`
```


PROBLEMS TERMINAL ... 2: Task - develop + - - -

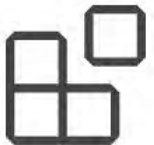
```
info [wdm]: Compiling...
DONE Compiled successfully in 26ms 3:57:58 PM
info [wdm]:
info [wdm]: Compiled successfully.
```

Ln 6, Col 21 Spaces: 2 UTF-8 LF JavaScript

 IntelliSense

 Run and Debug

 Built-in Git

 Extensions

첫 번째 리액트 앱

프로젝트 생성

☑ 스캐폴딩 툴, Scaffolding

- 기본 프로젝트 시작에 필요한 코드와 환경을 자동으로 세팅해주는 툴
- 프로젝트를 만들어주는 툴

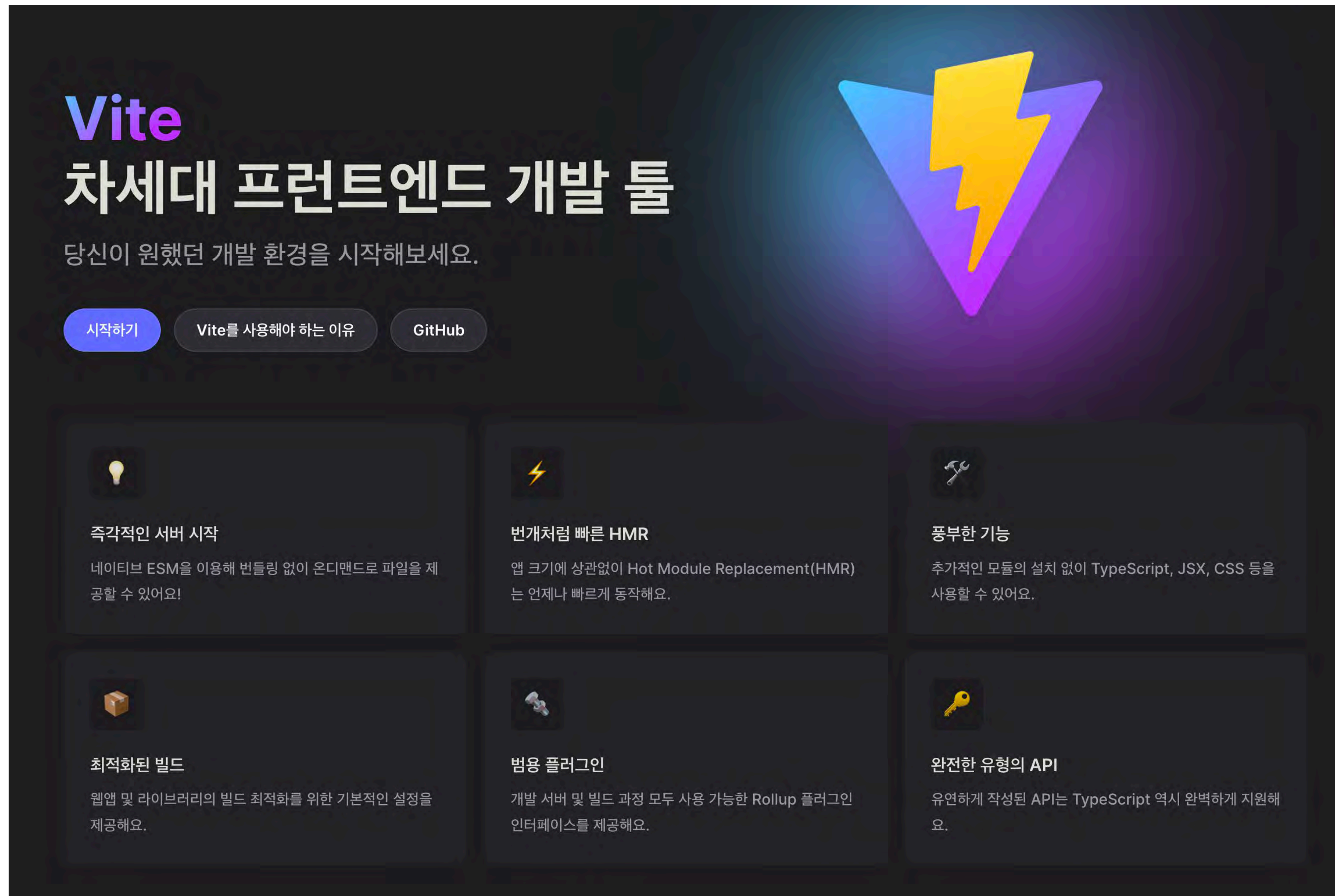
☑ 발전

- Angular 가 2016년 발표되면서 Angular-CLI 라는 스캐폴딩 툴 소개
- react - create-react-app 이라는 스캐폴딩 툴 발표
- Vue.js - VUE CLI 를 포함 -> Vue3 에서 스캐폴딩 툴 변경 (Vite)

☑ 현재 프로젝트는 모두 스캐폴딩을 통해서 프로젝트를 생성하고 개발 수행

- 장점 : 동일한 코드의 구조와 아키텍처로 개발 .. 가독성이 높아짐

vite - 프로젝트 스캐폴딩



The image shows the Vite website landing page. It features a dark background with a large, stylized Vite logo (a blue and yellow lightning bolt) on the right. The main heading is 'Vite' in a large, bold, purple font, followed by '차세대 프론트엔드 개발 툴' (Next-generation frontend development tool) in a large, bold, white font. Below this, a subtitle reads '당신이 원했던 개발 환경을 시작해보세요.' (Start the development environment you wanted). There are three buttons: '시작하기' (Get started), 'Vite를 사용해야 하는 이유' (Reasons to use Vite), and 'GitHub'. The page is divided into six sections, each with an icon and a title:

- 즉각적인 서버 시작** (Instant server start): 네이티브 ESM을 이용해 번들링 없이 온디맨드로 파일을 제공할 수 있어요! (You can provide files on-demand without bundling using native ESM!)
- 번개처럼 빠른 HMR** (Lightning-fast HMR): 앱 크기에 상관없이 Hot Module Replacement(HMR)는 언제나 빠르게 동작해요. (Hot Module Replacement(HMR) works fast regardless of app size.)
- 풍부한 기능** (Rich features): 추가적인 모듈의 설치 없이 TypeScript, JSX, CSS 등을 사용할 수 있어요. (You can use TypeScript, JSX, CSS, etc. without installing additional modules.)
- 최적화된 빌드** (Optimized build): 웹앱 및 라이브러리의 빌드 최적화를 위한 기본적인 설정을 제공해요. (We provide basic settings for optimizing the build of web apps and libraries.)
- 범용 플러그인** (Universal plugin): 개발 서버 및 빌드 과정 모두 사용 가능한 Rollup 플러그인 인터페이스를 제공해요. (We provide a Rollup plugin interface that can be used in both development server and build process.)
- 완전한 유형의 API** (Complete typed API): 유연하게 작성된 API는 TypeScript 역시 완벽하게 지원해요. (The flexibly written API is also fully supported by TypeScript.)

vite 로 리액트 프로젝트 만들기

☑ 프로젝트 스캐폴딩 - npm create vite

```
$ npm create vite@latest
```

```
// 이후 출력된 메시지에 의해 따라하면 됨
```

```
// react 프로젝트 생성 시
```

```
# npm 6.x
```

```
npm create vite@latest my-react-app --template react
```

```
# npm 7+, '--'를 반드시 붙여주세요
```

```
npm create vite@latest my-react-app -- --template react
```

```
# yarn
```

```
yarn create vite my-react-app --template react
```

```
# pnpm
```

```
pnpm create vite my-react-app --template react
```

vite 로 리액트 프로젝트 만들기

☑ 커맨드라인 인터페이스

```
{
  "scripts": {
    "dev": "vite", // 개발 서버를 실행합니다. (`vite dev` 또는 `vite serve`로도 시작이 가능합니다.)
    "build": "vite build", // 배포용 빌드 작업을 수행합니다.
    "preview": "vite preview" // 로컬에서 배포용 빌드에 대한 프리뷰 서버를 실행합니다.
  }
}
```


컴포넌트 만들기

☑ Hello World! 컴포넌트 만들기

- 각 컴포넌트는 별도의 파일로 만든다. (*.js 또는 *.jsx)
- 컴포넌트의 이름은 반드시 대문자로 시작되어야 한다.
- 컴포넌트 `return ()` 에는 하나의 태그만 위치

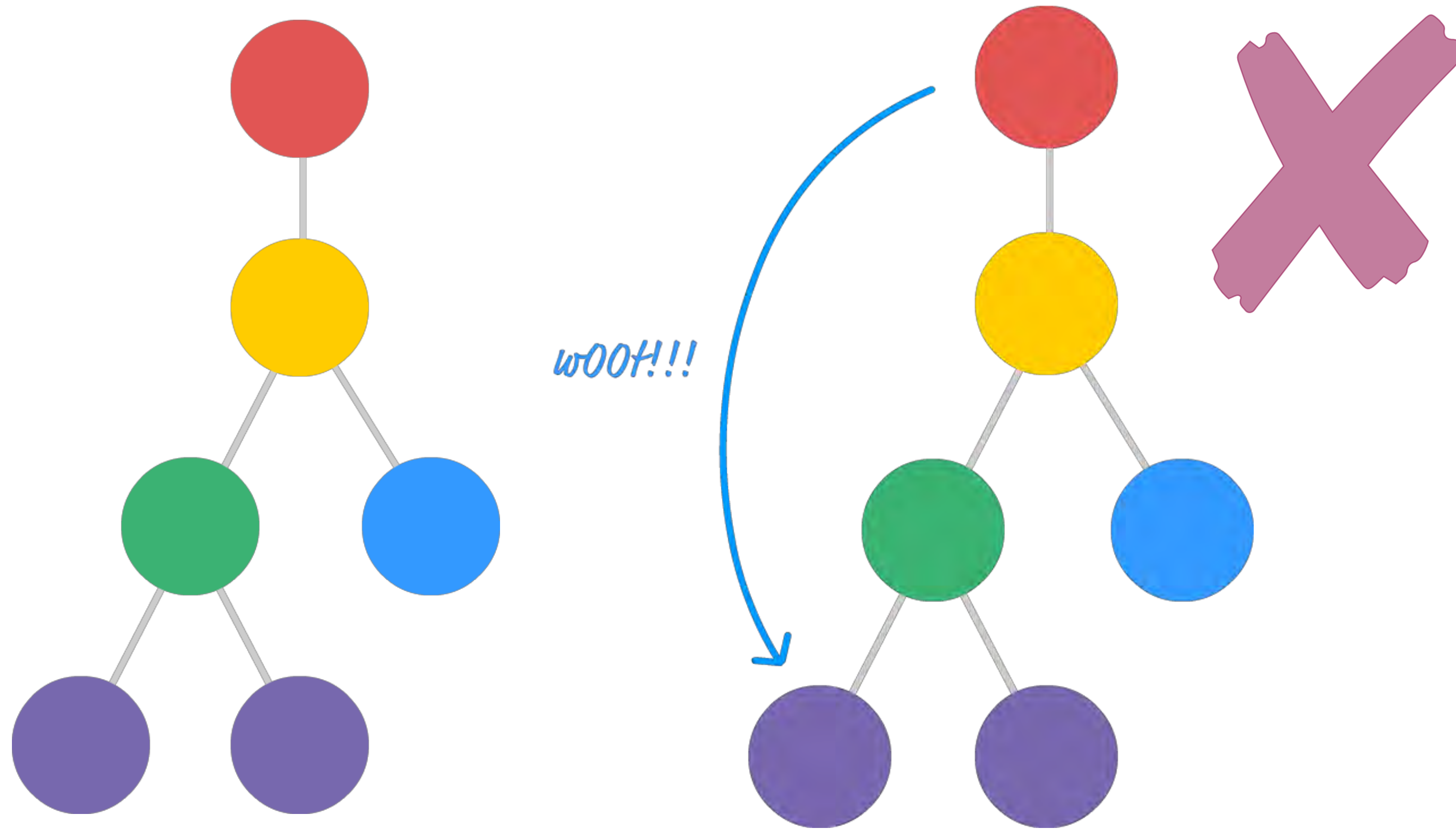
```
function App() {  
  return (  
    <h1>Hello, World</h1>  
  );  
}  
export default App;
```

부모 컴포넌트에서 자식 컴포넌트로 데이터 전달

PROPS

자식 컴포넌트에게 Props 전달

- ☑ 컴포넌트 간 속성 전달은 반드시 부모에서 자식 컴포넌트로만 가능



React - props

☑ **props** - 자식 컴포넌트에 데이터를 넘겨주자!

```
export default function Profile() {  
  return (  
    <Avatar  
      person={{ name: 'Lin Lanying', imageUrl: '1bX5QH6' }}  
      size={100}  
    />  
  );  
}
```

React - props

☑ 전달된 props 사용하기

```
function Avatar({ person, size }) {  
  // person 과 size 를 사용 가능 (destructuring)  
}
```

```
function Avatar(props) {  
  const person = props.person;  
  const size = props.size;  
  // props 로 받아서 사용가능  
}
```


React - props

☑ props 기본값 사용하기

```
function Avatar({ person, size = 100 }) {  
  
  // ..  
  
}
```

React - props

☑ 스프레드 연산자 사용하여 props 전달

```
function Profile({ person, size, isSepia, thickBorder }) {  
  return (  
    <div className="card">  
      <Avatar  
        person={person}  
        size={size}  
        isSepia={isSepia}  
        thickBorder={thickBorder}  
      />  
    </div>  
  );  
}
```

```
function Profile(props) {  
  return (  
    <div className="card">  
      <Avatar {...props} />  
    </div>  
  );  
}
```

React - children

☑ children 을 이용하여 콘텐츠를 전달

```
import Avatar from './Avatar.js';
```

```
function Card({ children }) {  
  return (  
    <div className="card">  
      {children}  
    </div>  
  );  
}
```

```
export default function Profile() {  
  return (  
    <Card>  
      <Avatar  
        size={100}  
        person={{  
          name: 'Katsuko Saruhashi',  
          imageUrl: 'YfeOqp2'  
        }}  
      />  
    </Card>  
  );  
}
```

if ~ else

조건부 렌더링

조건부 렌더링

☑ JSX 는 자바스크립트 코드

- if 문과 삼항 연산자 (? :) 을 이용해 조건부 렌더링이 가능

```
if (isPacked) {  
  return <li className="item">{name} ✓</li>;  
}  
return <li className="item">{name}</li>;
```

```
return (  
  <li className="item">  
    {isPacked ? name + ' ✓' : name}  
  </li>  
);
```


조건부 렌더링

☑ 논리 연산자 && 를 사용한 조건부 렌더링

```
return (  
  <li className="item">  
    {name} {isPacked && '✓'}  
  </li>  
);
```

- isPacked 가 true 이면 ✓ 사용됨, false 이면 사용되지 않음

map() 함수를 이용

리스트 렌더링

리스트 렌더링

☑ 배열(Arrays) 데이터를 반복적으로 렌더링

○ 배열의 map() 함수를 사용

```
const people = [  
  'Creola Katherine Johnson: mathematician',  
  'Mario José Molina-Pasquel Henríquez: chemist',  
  'Mohammad Abdus Salam: physicist',  
  'Percy Lavon Julian: chemist',  
  'Subrahmanyan Chandrasekhar: astrophysicist'  
];
```

```
const listItems = people.map(person => <li>{person}</li>);  
return <ul>{listItems}</ul>;
```

리스트 렌더링

- ☑ 순서대로 목록의 항목을 유지하기 위해 key 가 필요



- 각 리스트에 고유한 항목을 제공해야 함

```
<li key={person.id}>  
  <img src={getImageUrl(person)} />  
</li>
```


이벤트와 이벤트 핸들러

이벤트 처리

DOM onclick => JSX onClick

☑ JSX 에서 이벤트 처리는 함수를 사용

```
export default function Button() {  
  function handleClick() {  
    alert('You clicked me!');  
  }  
  
  return (  
    <button onClick={handleClick}>  
      Click me  
    </button>  
  );  
}
```



// 간결하게 화살표함수 사용

```
<button onClick={() => {  
  alert('You clicked me!');  
}}>
```

이벤트 핸들러를 props 로 전달

☑ props 로 이벤트 핸들러(함수)를 전달

```
function Button({ onClick, children }) {  
  return (  
    <button onClick={onClick}>  
      {children}  
    </button>  
  );  
}
```

```
function PlayButton({ movieName }) {  
  function handlePlayClick() {  
    alert(`Playing ${movieName}!`);  
  }  
  return (  
    <Button onClick={handlePlayClick}>  
      Play "{movieName}"  
    </Button>  
  );  
}
```

이벤트 전파 - bubbling, propagation

- ☑ 이벤트는 발생하는 요소에서 위로 전파됨

```
export default function Toolbar() {  
  return (  
    <div className="Toolbar" onClick={() => {  
      alert('You clicked on the toolbar!');  
    }}>  
      <button onClick={() => alert('Playing!')}>  
        Play Movie  
      </button>  
      <button onClick={() => alert('Uploading!')}>  
        Upload Image  
      </button>  
    </div>  
  );  
}
```


이벤트 전파 방지

- ☑ 이벤트 핸들러는 이벤트 객체를 유일한 파라미터로 받음
 - e.stopPropagation() 을 사용하여 이벤트 전파 방지 가능

- ☑ 이벤트 전파 중지와 관계없이 이벤트 캡처 가능

```
<div onClickCapture={() => { /* 우선순위를 가짐 */ }}>  
  <button onClick={e => e.stopPropagation()} />  
  <button onClick={e => e.stopPropagation()} />  
</div>
```

DOM 직접 조작 - useRef

- ☑ 상태 변경 트리거에 의해 DOM 이 변경되는 것이 React 의 기본 기능
 - 그러나 DOM 을 직접 조작해야 하는 경우가 생김
 - useRef 를 사용하여 DOM 에 직접 접근 및 조작

```
import { useRef } from 'react';  
// ref 선언  
const inputRef = useRef(null);  
// 특정 요소에 ref 를 전달  
<div ref={inputRef}>  
// DOM 에 직접 접근 및 조작  
inputRef.current.focus();
```

React Hook

리액트 후크

리액트 Hook

- ☑ 2018년 16.8.0부터 공식적으로 지원
- ☑ 리액트 16.8 이전 리액트 컴포넌트 작성 방식
 - 클래스 기반의 컴포넌트 - 생명주기 메소드와 state를 가질 수 있음

리액트 Hook 을 사용하면 모든 컴포넌트를 함수형 컴포넌트로 작성 가능

리액트 Hook - 클래스 컴포넌트 단점

❑ 복잡성:

- 자바스크립트에서 클래스와 관련된 특정 지식이 필요
- 이는 학습곡선을 높게 만들고, 코드의 가독성을 낮춥니다.

❑ 재사용성:

- 복잡한 패턴(예: render props, higher order components)을 사용해야 함
- 이런 패턴들은 코드를 어렵게 만들고 컴포넌트 재구성을 어렵게 함

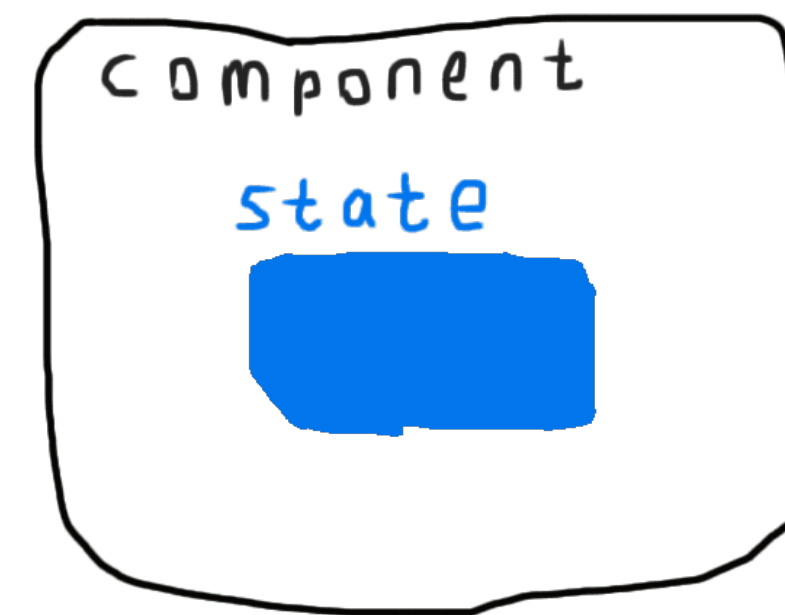
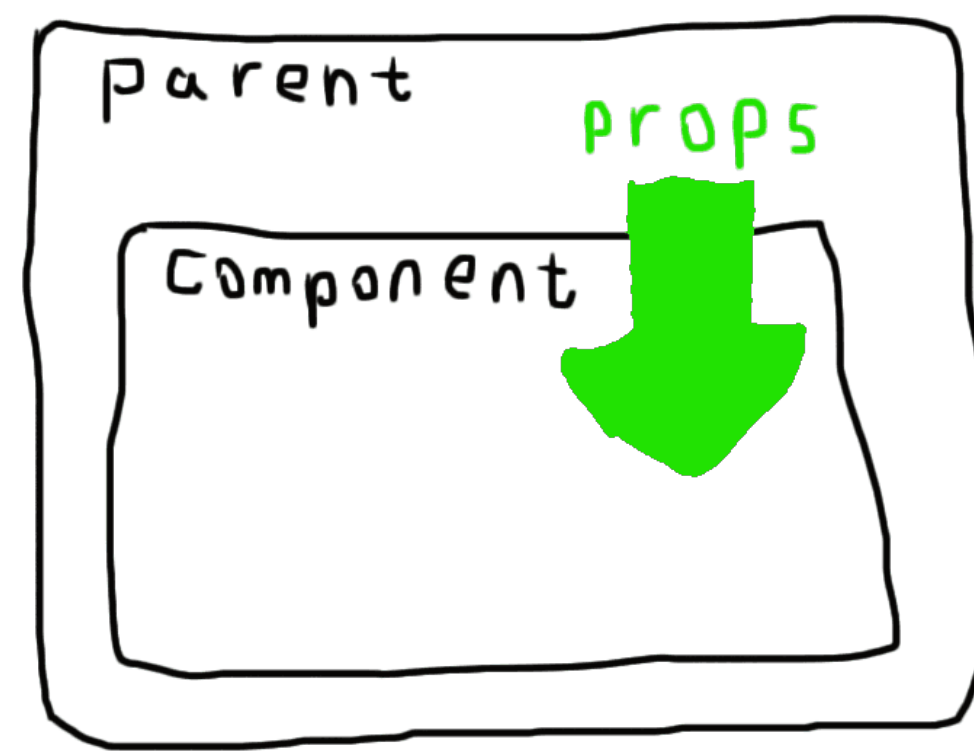
❑ 버그:

- 생명주기 메소드에서 발생하는 버그.
- componentDidMount 및 componentDidUpdate에서 동일한 로직을 반복적으로 쓰는 경우, 이 두 메소드 사이에서 로직이 일치하지 않으면 버그가 발생할 수 있음

리액트 Hook - 사용의 의미

- ☑ 함수형 프로그래밍과 디자인 패턴을 통한 간결성과 가독성 제공
- ☑ 로직의 재사용과 캡슐화
- ☑ 컴포넌트 생명주기와 상태 관리의 통합

리액트의 사용자 경험을 크게 향상시키고
더욱 강력한 도구를 제공하여
프론트엔드 개발의 효율성과 품질을 높임



자체 데이터 포함 컴포넌트

STATE

State, 상태

- ☑ 컴포넌트는 화면이고 화면은 데이터를 사용한다.
- ☑ 데이터는 원본이 있고, 사본이 있다.
 - 원본 : State, 사본 : Props
- ☑ 원본은 변경 가능(mutable), 사본은 변경 불가능(immutable)
 - 원본이 변경되면 사본이 같이 변한다.

데이터가 변하면 해당 데이터를 사용하는 화면이 즉각 변한다

State 가 변하면 화면이 다시 그려지는 원리

☑ 렌더링 트리거

- 최초 화면이 로딩 될 때
- State 값이 변경되었을 때

☑ 렌더링 수행

- 변경될 태그들을 계산 -> 실제 화면에 반영되지 않음

☑ DOM 커밋

- 실제 화면에 반영되어 화면에 그려짐

useState() Hook

- ☑ state 값 관리를 위한 useState()
 - 함수형 컴포넌트로 state 관리 가능 (react 16.8 에서 Hook 도입)
- ☑ `const [number, setNumber] = useState(0);`
 - `number` : state 이름
 - `setNumber` : `number` state 값 변경 함수
 - `0` : `number` state 의 기본 값

리액트에서의 상태(State) 관리

☑ 리액트 훅(hook)을 통해 상태를 관리

○ useState()

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

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```


참고 : 선언적 프로그래밍?

☑ 명령형 프로그래밍 방식

- Imperative Programming
- 데이터가 변경되면, 변경된 데이터로 화면(DOM)을 직접 변경한다.
- programmatic

☑ 선언적 프로그래밍 방식

- Declarative Programming
- 데이터가 변경되면, 변경된 데이터를 사용하는 화면이 자동적으로 변경된다.
- reactive

useEffect()

Component Lifecycle

컴포넌트 라이프 사이클과 EFFECT 훅

컴포넌트가 업데이트 될 때

- ☑ Props 가 바뀔 때
- ☑ State 가 바뀔 때
- ☑ 부모 컴포넌트가 리렌더링 될 때

컴포넌트는 리액트에 의해 자동으로 관리된다.

메모리에 로딩(탄생), 화면에 렌더링, 메모리 제거(죽음)

개발자가 직접 제어할 수 없다

Effect? (or Side Effect?)

☑ React 의 주요 역할

- UI 렌더링, 사용자 입력 처리
- JSX 렌더링

☑ Side Effect 란?

- 리액트 주요 역할 외 모든 작업
- 브라우저 DB에 데이터 저장
- 백엔드 서버에 HTTP 요청으로 데이터 전송
- 타이머 설정 등..

useEffect() Hook

- ☑ SideEffect 처리를 위한 useEffect() 훅
 - 클래스 컴포넌트의 라이프 사이클에 대응하기 위해 만들어 짐
 - 컴포넌트가 마운트 / 언마운트 되었을 때 수행
 - 특정 state/props 가 변경될 때 특정 작업 처리 가능

useEffect() Hook

☑ `useEffect(() => { ... }, [])`

- 첫 번째 파라미터 : 수행되는 코드
 - 함수를 반환 가능 : `cleanup` 함수
- 두 번째 파라미터 : 의존값, `deps`
 - `[]` 빈 배열 전달시 컴포넌트 최초 마운트 시 코드 수행
 - `deps`에 state 값 추가 시 state 값 변경시 코드 수행
 - 파라미터가 없을 시 컴포넌트 업데이트 시 매번 수행

useMemo Hook

- ☑️ useMemo 사용하여 연산한 값 재사용하기
 - 함수 컴포넌트 내의 지역변수는 컴포넌트 리렌더링 후 초기화 됨
 - 특정 데이터를 사용할 때 매번 연산하는 수고를 덜어줌
- ☑️ `const count = useMemo(() => countProducts(products), [products]);`
 - products 값이 변하지 않으면 countProducts 함수를 수행하지 않음
 - 기존의 count 값을 그대로 사용

React.memo

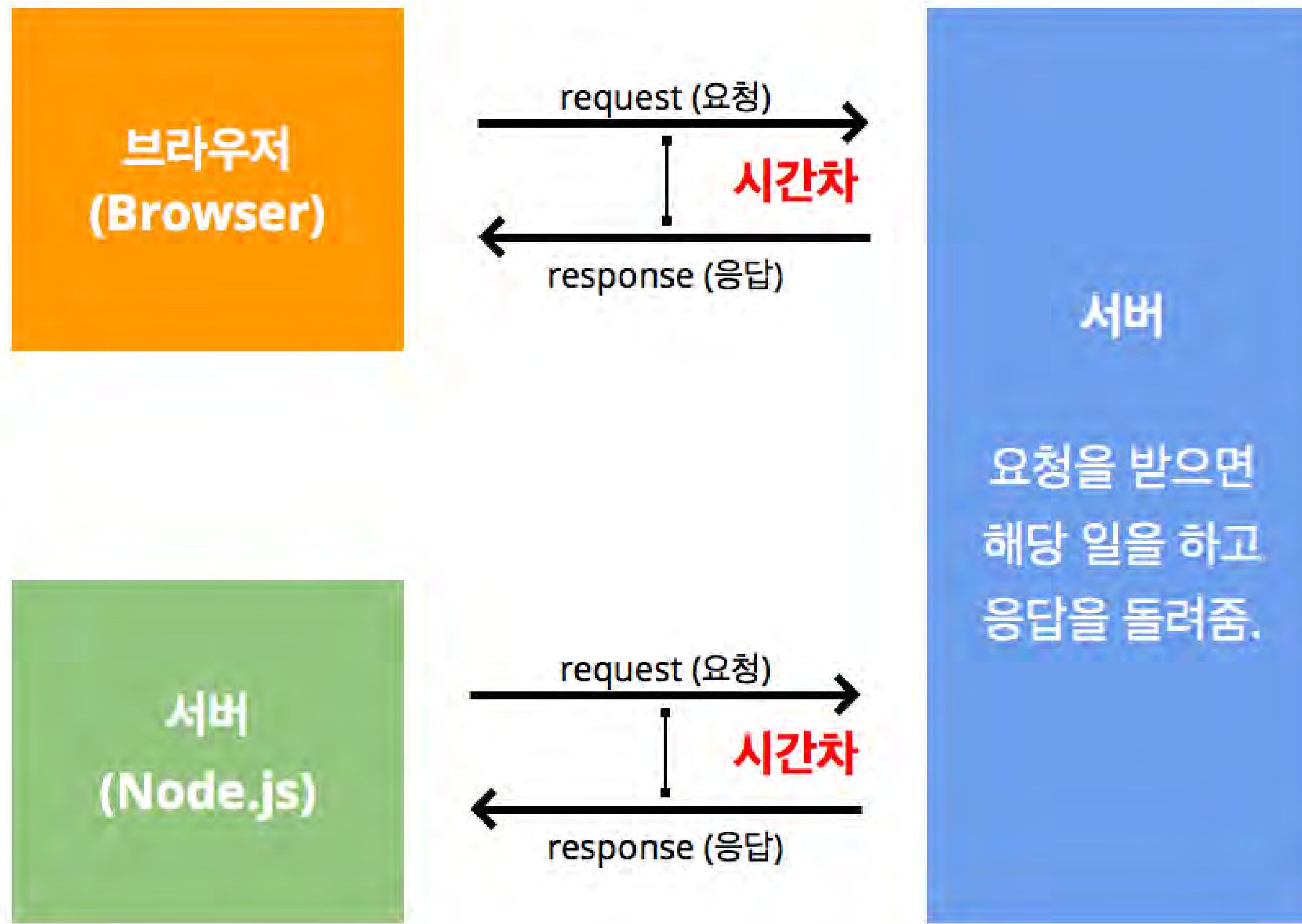
- ☑ React.memo 를 사용하여 컴포넌트 리렌더링 방지
 - 컴포넌트의 props 가 바뀌지 않았다면, 리렌더링 방지
 - 성능 최적화
 - 컴포넌트를 export 시 React.memo 로 감싸주면 됨
- ☑ export default React.memo(MyComponent)
 - MyComponent 는 props 가 바뀌지 않으면 리렌더링 되지 않음

API 서비스

axios 라이브러리 사용

데이터 FETCH - 서버에서 데이터 가져오기

요청과 응답



API 요청 방식

☑ REST API 요청 방식에 따른 작업

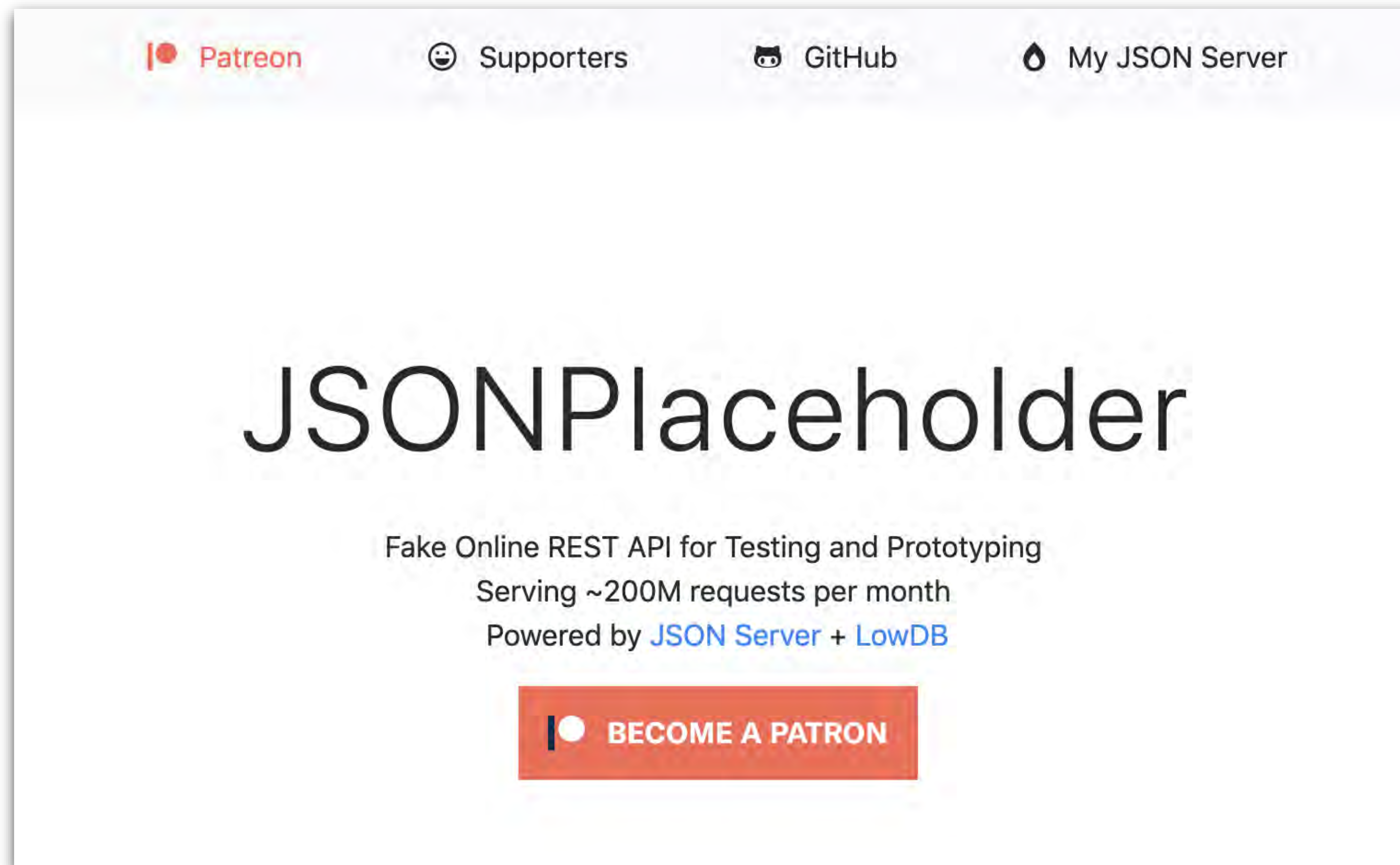
- GET : 데이터 조회
- POST : 데이터 등록
- PUT : 데이터 수정
- DELETE : 데이터 삭제

☑ axios 사용법

```
import axios from 'axios';  
axios.get('/users/1');
```

jsonplaceholder :: fake API(json) 서비스

<https://jsonplaceholder.typicode.com/>



Data Fetching - useEffect() 사용 예제

- ☑ useState() 와 useEffect() 사용하여 데이터 로딩
 - useState() 로 상태 관리
 - useEffect() 를 사용 컴포넌트가 렌더링 직후 시점에 요청을 시작

```
const [users, setUsers] = useState([]); // [] 초기값으로 users 데이터가 비어있음을 의미

useEffect(async () => {
  try {
    const response = await axios.get('https://jsonplaceholder.typicode.com/users');
    setUsers(response.data); // 데이터는 response.data 안에 들어있습니다.
  } catch (e) {
    console.log(e); // 에러 발생시 처리
  }, []); // []는 컴포넌트가 마운트될 때만 effect를 실행하겠다는 것을 의미합니다.
```

Data Fetching - useReducer() 사용 예제

- ☑ useReducer 로 요청 상태 관리
 - 액션 : LOADING, SUCCESS, ERROR
- ☑ 요청에 대한 상태 3 가지
 - 요청의 결과 - `const [users, setUsers] = useState(null);`
 - 로딩 상태 - `const [loading, setLoading] = useState(false);`
 - 에러 - `const [error, setError] = useState(null);`

reducer란 무엇인가?

☑ reducer 함수는

- 현재 상태와 액션 객체를 파라미터로 받아 새로운 상태를 반환
- reducer 에서 반환하는 상태는 곧 컴포넌트의 새로운 상태
- action 은 업데이트를 위한 정보를 갖고 있음

```
function reducer(state, action) {  
  // 새로운 상태를 만드는 로직  
  // const nextState = ...  
  return nextState;  
}
```

reducer란 무엇인가?

- ☑ reducer 함수의 action 예시
 - action 객체의 형태는 자유임

```
// 카운터에 1을 더하는 액션
{
  type: 'INCREMENT'
}
// 카운터에 1을 빼는 액션
{
  type: 'DECREMENT'
}
```

```
// input 값을 바꾸는 액션
{
  type: 'CHANGE_INPUT',
  key: 'email',
  value: 'tester@react.com'
}
// 새 할 일을 등록하는 액션
{
  type: 'ADD_TODO',
  todo: {
    id: 1,
    text: 'useReducer 배우기',
    done: false,
  }
}
```

useReducer Hook 예제

☑ useReducer 란?

- state 업데이트 로직은 컴포넌트 내부에서 이루어짐
 - useState 혹은 통해 상태 업데이트
- state 값 업데이트 하는 다른 방법 - useReducer
- 컴포넌트의 state 업데이트 로직을 컴포넌트 외부로 별도로 분리 가능

☑ `const [state, dispatch] = useReducer(reducer, initialState);`

- reducer 함수를 통해 업데이트 로직 분리

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'INCREMENT':
      return {count: state.count + 1};
    case 'DECREMENT':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'DECREMENT'})}>-
    </button>
      <button onClick={() => dispatch({type: 'INCREMENT'})}>+</button>
    </>
  );
}
```

useReducer Hook 예제

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'INCREAMENT':
      return {count: state.count + 1};
    case 'DECREAMENT':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}
```

```
function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'DECREAMENT'})}>-</button>
      <button onClick={() => dispatch({type: 'INCREAMENT'})}>+</button>
    </>
  );
}
```

서버에서 데이터 받아오기 - useReducer() 사용 예제

☑ useReducer 로 요청 상태 관리 - 액션 : LOADING, SUCCESS, ERROR

```
function reducer(state, action) {  
  switch (action.type) {  
    case 'LOADING':  
      return {  
        loading: true, data: null, error: null  
      };  
    case 'SUCCESS':  
      return {  
        loading: false, data: action.data, error: null  
      };  
    case 'ERROR':  
      return {  
        loading: false, data: null, error: action.error  
      };  
    default:  
      throw new Error(`Unhandled action type: ${action.type}`);  
  }  
}
```


Data Fetching - useReducer() 사용 예제

☑ useReducer 로 요청 상태 관리

```
const [state, dispatch] = useReducer(reducer, {  
  loading: false,  
  data: null,  
  error: null  
});
```

○ 상태 변경 시 dispatch 사용

```
const fetchUsers = async () => {  
  dispatch({ type: 'LOADING' });  
  try {  
    const response = await axios.get(  
      'https://jsonplaceholder.typicode.com/users'  
    );  
    dispatch({ type: 'SUCCESS', data: response.data });  
  } catch (e) {  
    dispatch({ type: 'ERROR', error: e });  
  }  
};
```

useReducer Hook

☑ useReducer 란?

- state 업데이트 로직은 컴포넌트 내부에서 이루어짐
 - useState 혹은 통해 상태 업데이트
- state 값 업데이트 하는 다른 방법 - useReducer
- 컴포넌트의 state 업데이트 로직을 컴포넌트 외부로 별도로 분리 가능

☑ `const [state, dispatch] = useReducer(reducer, initialState);`

- reducer 함수를 통해 업데이트 로직 분리

커스텀 Hook 만들기

☑ useAsync() Hook

- useAsync.js 파일에 생성

☑ `const [state, fetchData] = useAsync(callback(), deps)`

- `callback()` : API 요청을 시작하는 함수
- `deps` : `useEffect` 의 `deps` 로 설정
- 반환값 : `state` 와 요청을 할 수 있는 `fetchData` 함수

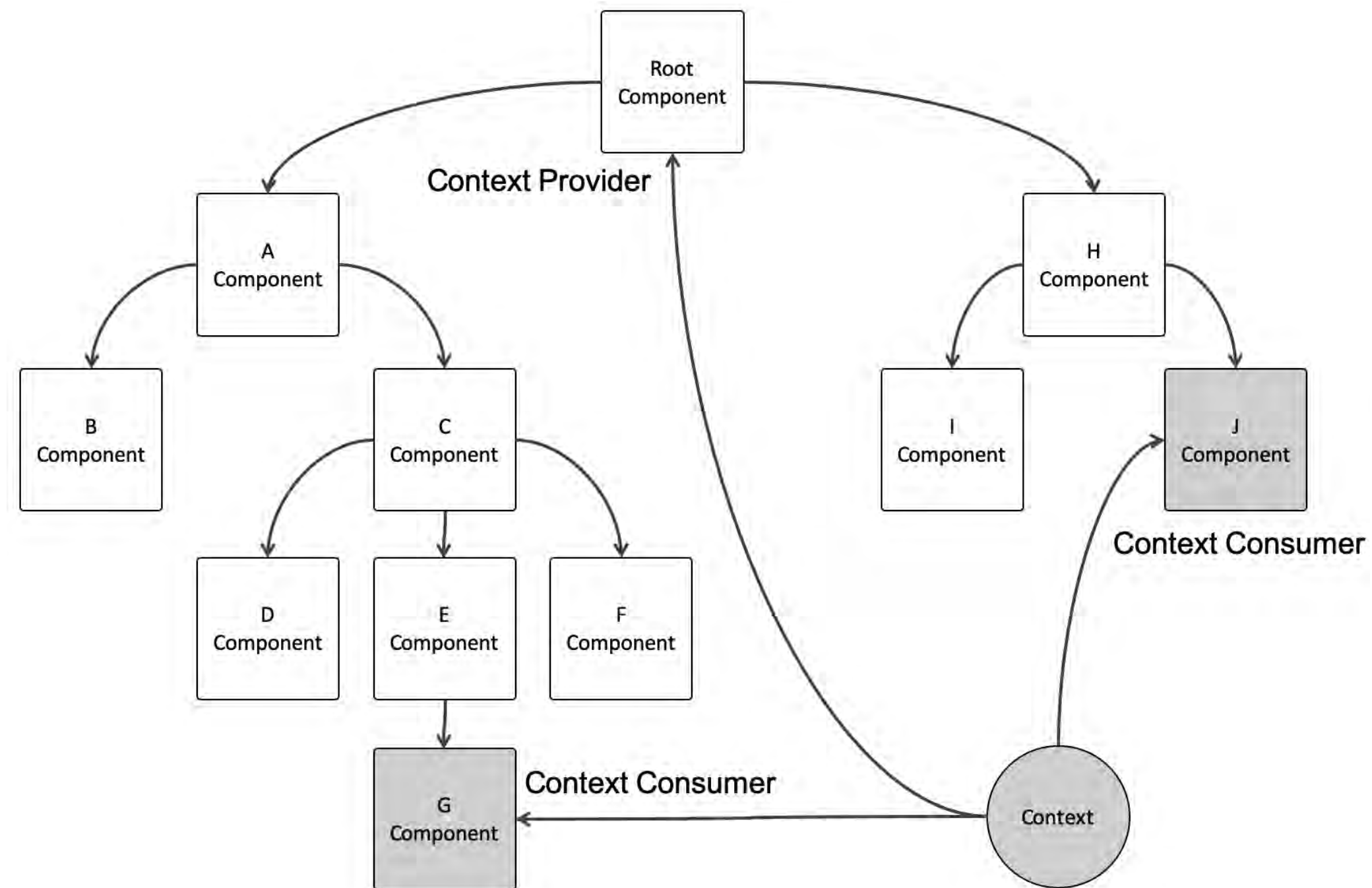
전역 데이터 관리하기

CONTEXT

Context

☑ Context 란?

- 컴포넌트 간 간격에 제약받지 않고 데이터를 전달 가능
- props 와 state 는 부모와 자식간 으로만 이동이 가능



Context 주요 API

☑ Context 란?

- 전역 데이터를 담고 있는 하나의 저장공간
- createContext 함수를 사용하여 생성
- 전역적으로 데이터를 공유하도록 나온 개념
 - 로그인 데이터, 설정파일, 테마, 언어 ..

```
import { createContext } from "react";  
  
const LangContext = createContext("en");
```

Context 생성

☑ 카운터 Context 생성

○ 전역으로 저장할 데이터 초기화

```
export const CountContext = createContext({  
  count: 0,  
  increaseCount: () => {},  
  decreaseCount: () => {},  
});
```

○ 컴포넌트에서 변경 가능한 값을 다루기 위해 state 값을 사용

```
const [count, setCount] = useState(0);  
const increaseCount = () => {  
  setCount(prev => prev + 1);  
}  
const decreaseCount = () => {  
  setCount(prev => prev - 1);  
}
```


Context 생성

☑ 카운터 Context 생성

○ state 를 Context 의 Provider 에 제공

```
return (  
  <CountContext.Provider  
    value={{  
      count,  
      increaseCount,  
      decreaseCount,  
    }}>  
    {children}  
  </CountContext.Provider>  
);
```

Context 사용

- ☑ Context 사용하기 위해 공통 부모 컴포넌트에 Provider 제공

```
function App() {  
  return (  
    <CountProvider>  
      <CountLabel></CountLabel>  
      <CountButton></CountButton>  
    </CountProvider>  
  )  
}
```

- ☑ Consumer

- useContext() 혹은 사용

```
const {count} = useContext(CountContext);
```

```
const {increaseCount, decreaseCount} = useContext(CountContext);
```

react router

라우팅

라우팅이란?

- ☑ 동적 웹앱에서 가장 중요한 부분 중 하나
 - 사용자가 원하는 위치(페이지)로 이동
 - 메뉴와 관련이 있음
 - 예) website.com/about URL 요청에 정보 페이지로 라우팅 됨
 - SPA 라우팅은 거의 동일 (Angular, React, Vue ..)
 - HTTP 요청을 처리하는 코드에 연결하는 방법을 결정하는 매칭 매커니즘

URL을 기반으로 원하는 리소스(페이지)에 연결 주는 기능

라우팅 : React Router

- ☑ React Router - <https://reactrouter.com/>.
 - 다양한 기능으로 페이지를 새로 고침 필요없이 전환
 - npm 으로 설치

☑ React Router 설정

```
$ npm install react-router-dom
```

```
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";

const root = ReactDOM.createRoot(
  document.getElementById("root")
);
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

Navigation

☑ Link

- <a> 태그와 유사
- 메뉴로 사용 가능

☑ useNavigate

- 프로그래머틱 하게 사용

```
function Home() {  
  return (  
    <div>  
      <h1>Home</h1>  
      <nav>  
        <Link to="/">Home</Link> |{" "  
        <Link to="about">About</Link>  
      </nav>  
    </div>  
  );  
}
```

```
function Invoices({checkValidation}) {  
  let navigate = useNavigate();  
  const checkLogin = () => {  
    if(checkValidation){  
      navigate(`/main`)  
    }  
  }  
  return (  
    ...  
    <div>  
      <LoginBtn onClick={checkLogin}/>  
    </div>  
  );  
}
```

URL 파라미터 읽어오기

- ☑ 경로에 파라미터 전달 가능
 - useParams() 로 경로 읽기 가능

```
function App() {  
  return (  
    <Routes>  
      <Route  
        path="invoices/:invoiceId"  
        element={<Invoice />}  
      />  
    </Routes>  
  );  
}
```

```
function Invoice() {  
  let params = useParams();  
  return <h1>Invoice {params.invoiceId}</h1>;  
}
```

중첩된 라우트, Nested Routes

- ☑ 리액트 라우트에서 가장 강력한 기능 중 하나

```
function App() {  
  return (  
    <Routes>  
      <Route path="/invoices" element={<Invoices />}>  
        <Route path=":invoiceId" element={<Invoice />} />  
        <Route path="sent" element={<SentInvoices />} />  
      </Route>  
    </Routes>  
  );  
}
```

<Outlet /> 사용

- /invoices
- /invoices/:invoiceId
- /invoices/sent

“Not Found” 경로

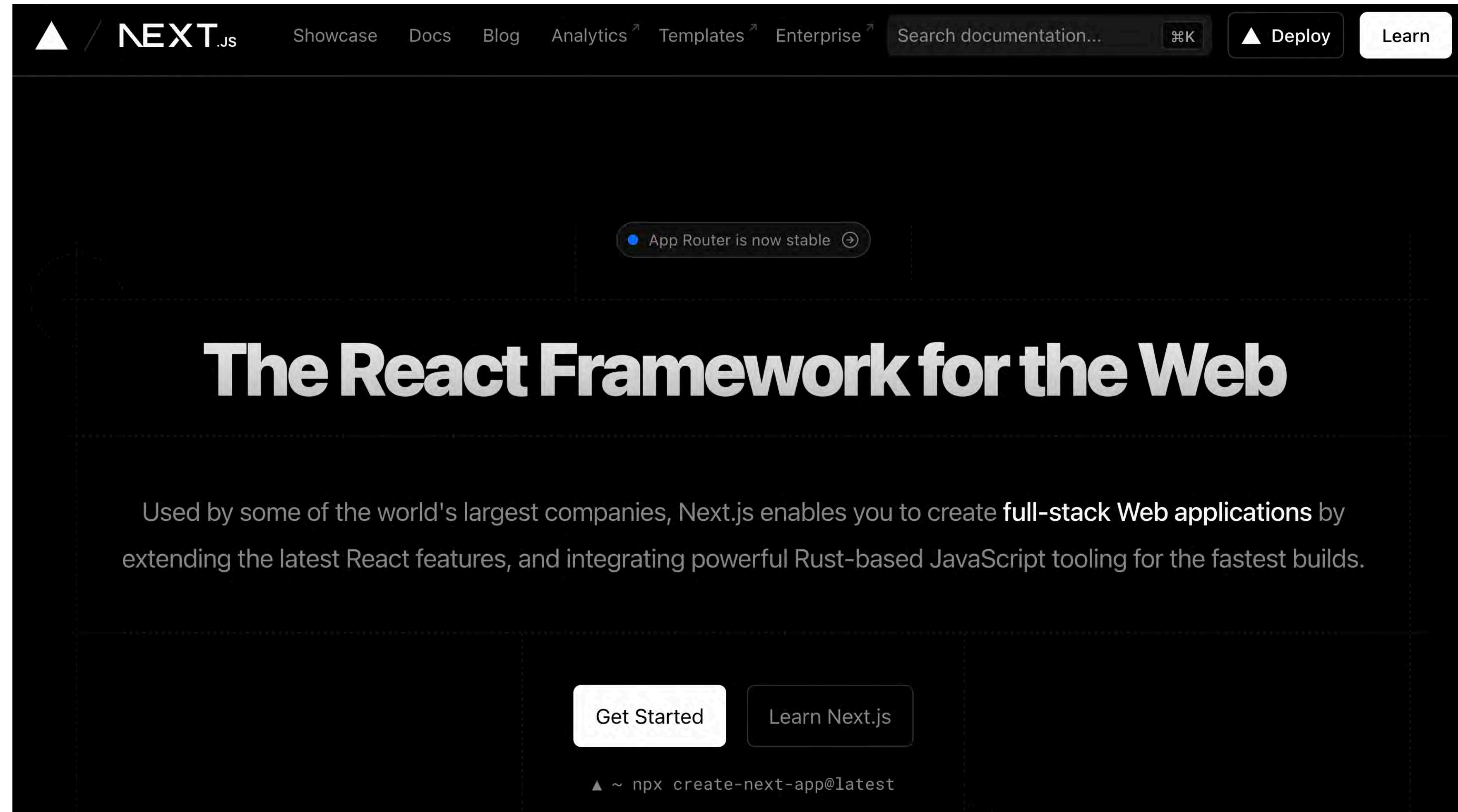
☑ path=“*”

- 어떤 route 에도 url 에 맞지 않을 때
- 가장 낮은 순위를 가지기 때문에 다른 라우트가 맞지 않을 때 선택

```
function App() {  
  return (  
    <Routes>  
      <Route path="/" element={<Home />} />  
      <Route path="dashboard" element={<Dashboard />} />  
      <Route path="*" element={<NotFound />} />  
    </Routes>  
  );  
}
```

NEXT.JS

Next.js - 리액트 프레임워크



Next.js란?

- ☑ React 기반의 오픈 소스 프레임워크로, 풀스택 웹 애플리케이션을 구축하는 데 사용
 - 2016년에 Vercel이라는 회사(당시의 Zeit)에 의해 출시
- ☑ 주요 기능
 - 서버 사이드 렌더링 (SSR)
 - 정적 사이트 생성 (SSG)
 - 핫 모듈 교체 (HMR)
 - 파일기반 라우팅

리액트 영화 앱

리액트 영화 앱 - 소개

 리액트 무비앱

현재 상영 중 인기 상영작 높은 평점

 Search...



트랜스포머: 비스트의 서막

전 우주의 행성을 집어삼키는 절대자, '유니크론'의 부하 '스커지'는 '테러콘'들을 이끌고 지구에 당도한다. 그에 맞서기 위해 지구에 정체를 숨기고 있던 트랜스포머 '오토부' 구단이 무섭게 드러내고 또 다른 트랜스포머



세인트 세이아: 더 비기닝

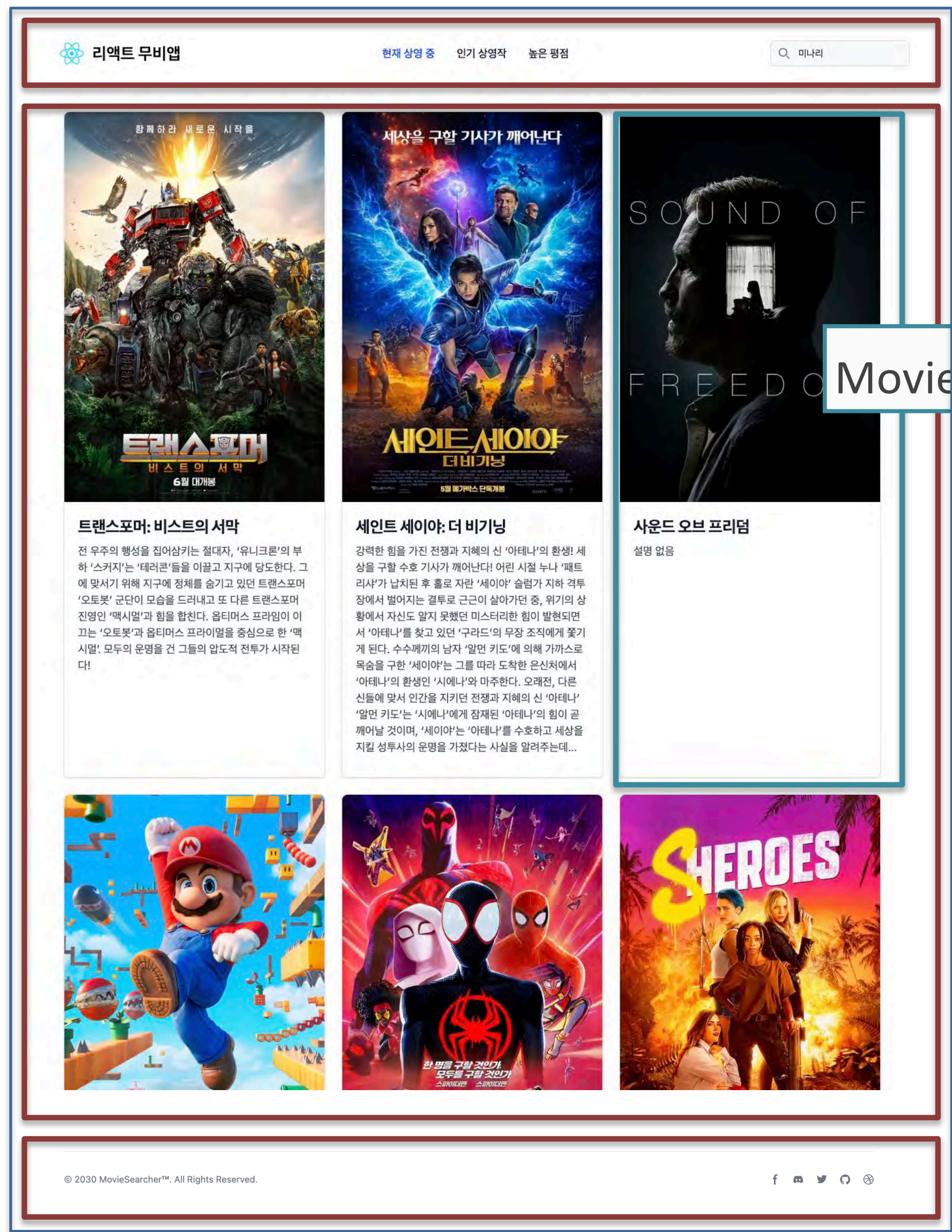
강력한 힘을 가진 전쟁과 지혜의 신 '아테나'의 환생! 세상을 구할 수호 기사가 깨어난다! 어린 시절 누나 '패트리샤'가 납치된 후 홀로 자란 '세이아' 슬럼가 지하 격투장에서 벌어지는 격투로 구구이 삭아가던 중 위기의 사



사운드 오브 프리덤

설명 없음

리액트 영화 앱 - 컴포넌트 구조



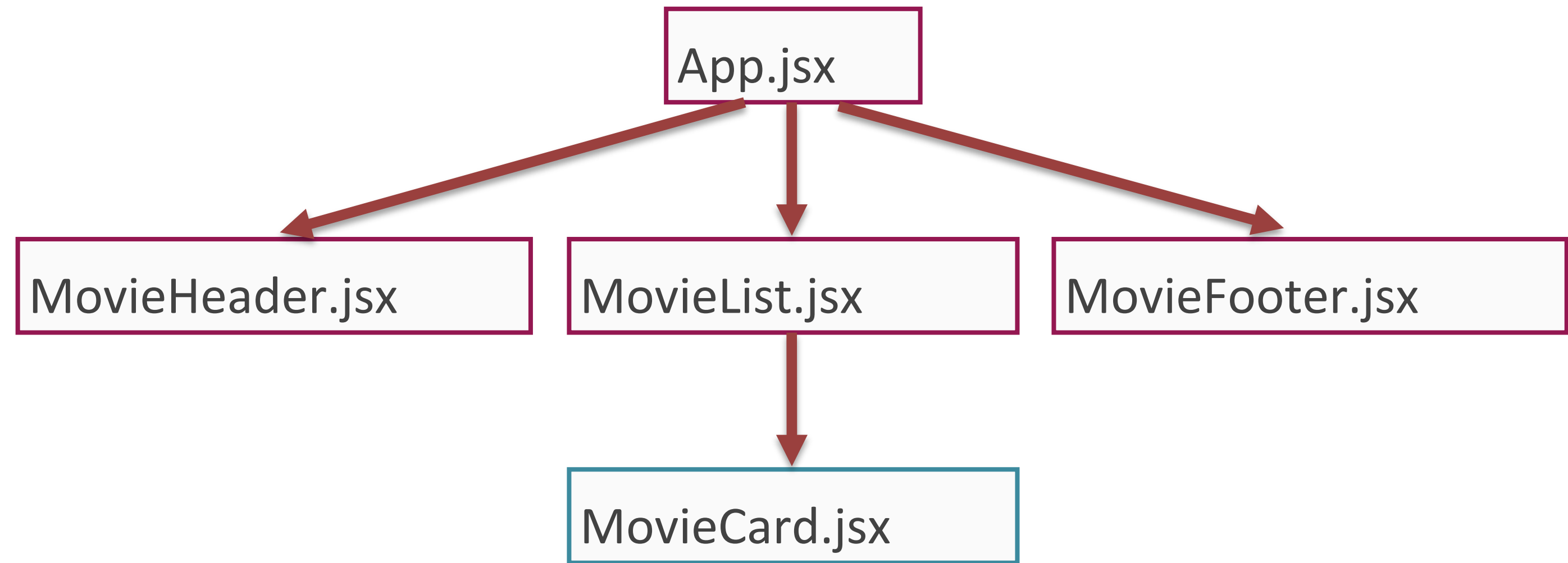
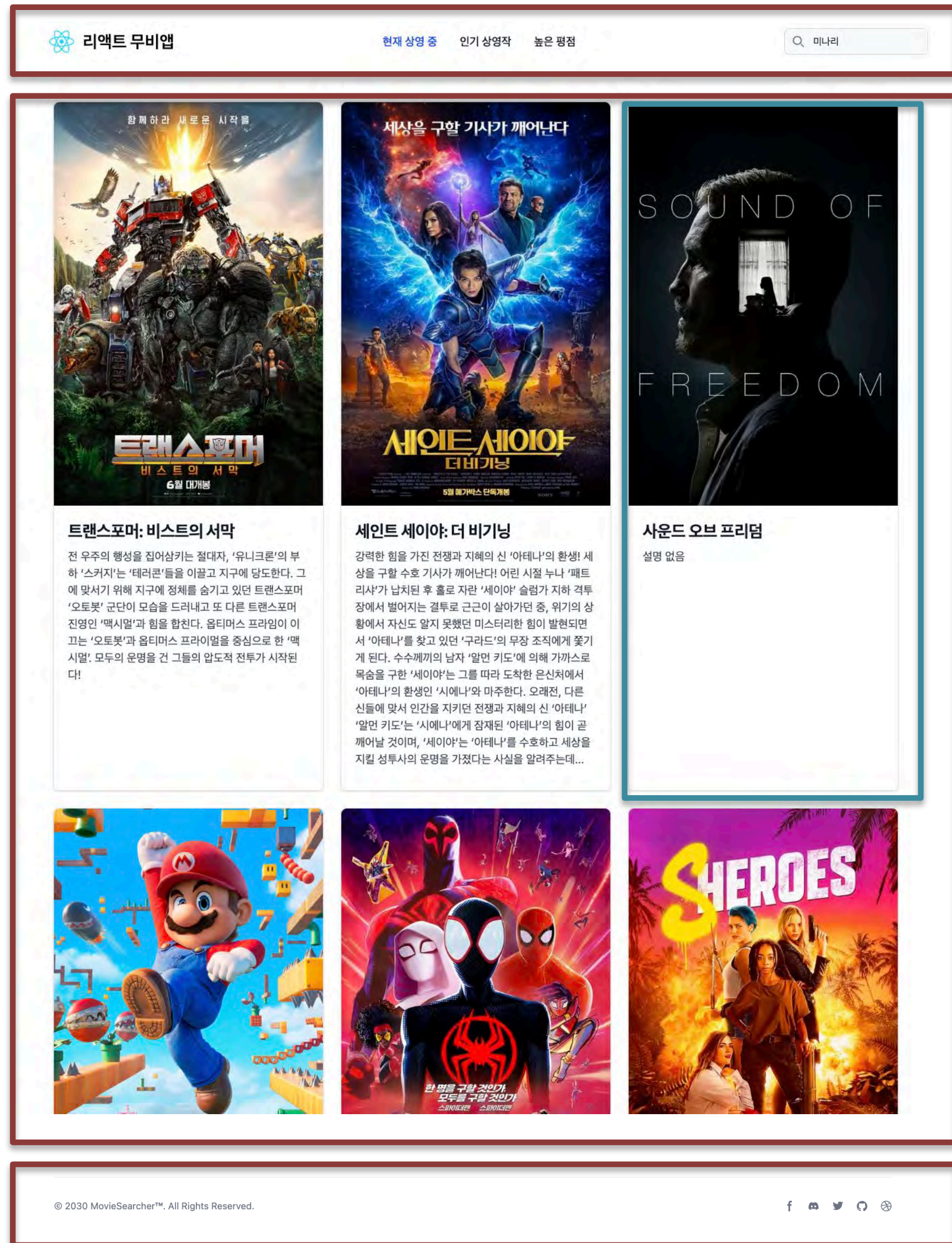
MovieHeader.jsx

MovieCard.jsx

MovieList.jsx

MovieFooter.jsx

리액트 영화 앱



리액트 영화 앱 - 앱 구성

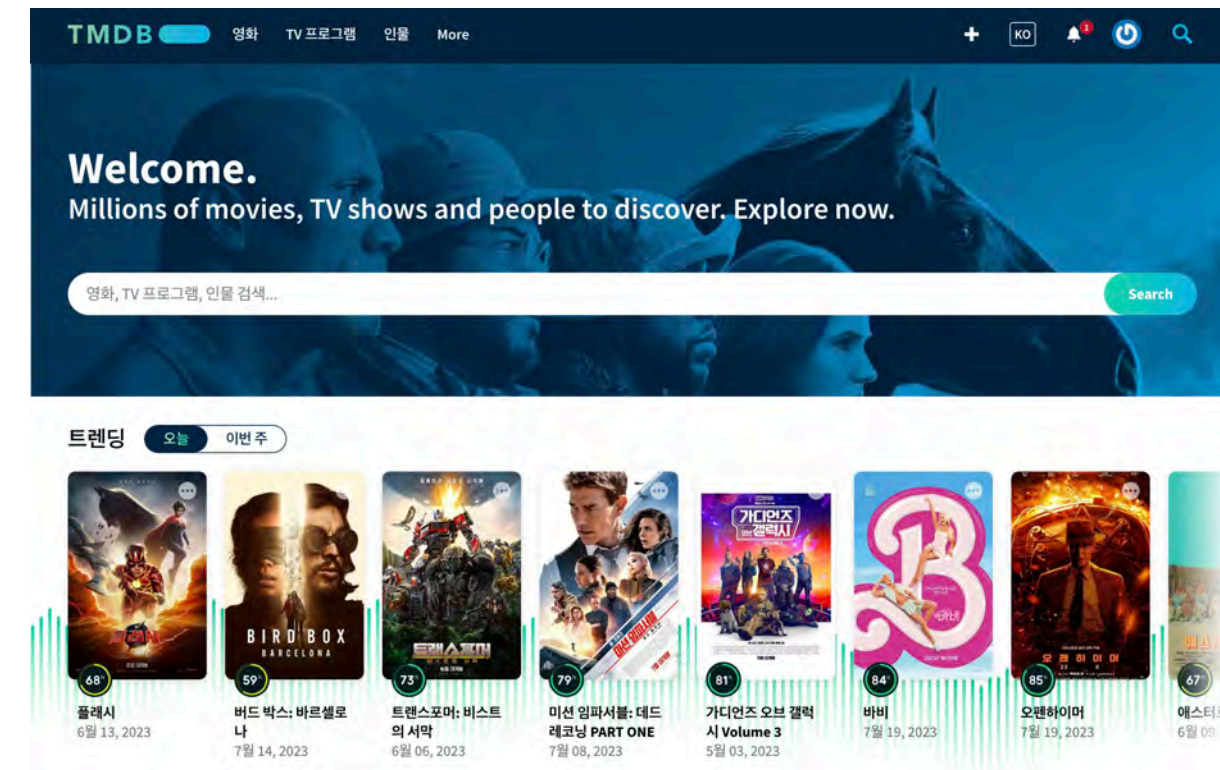
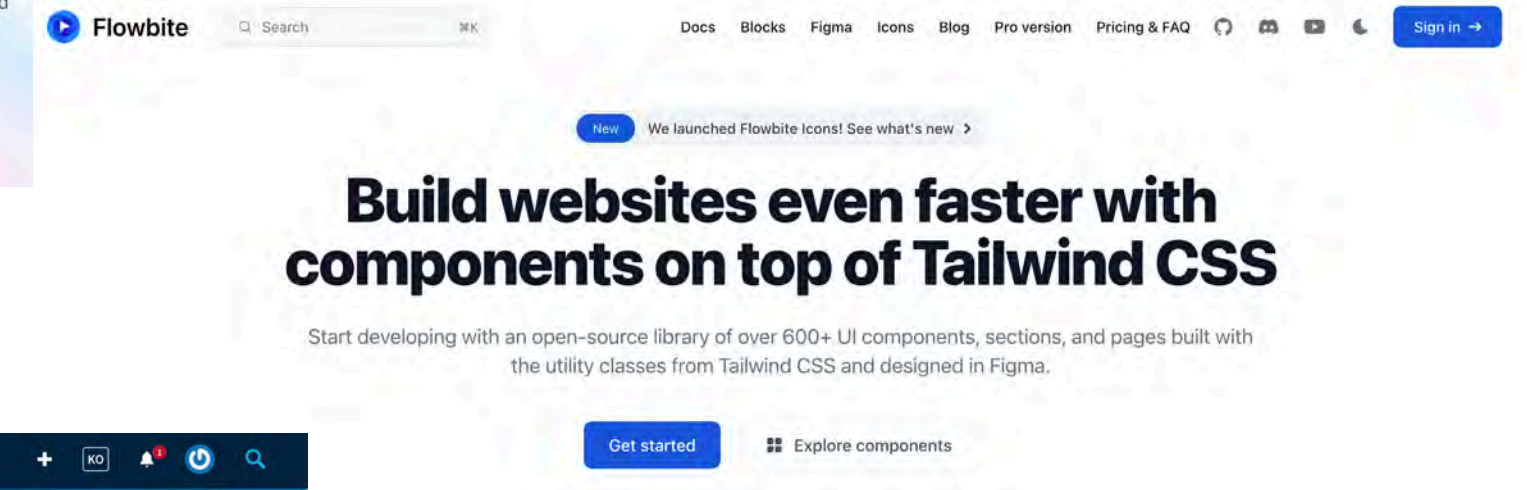
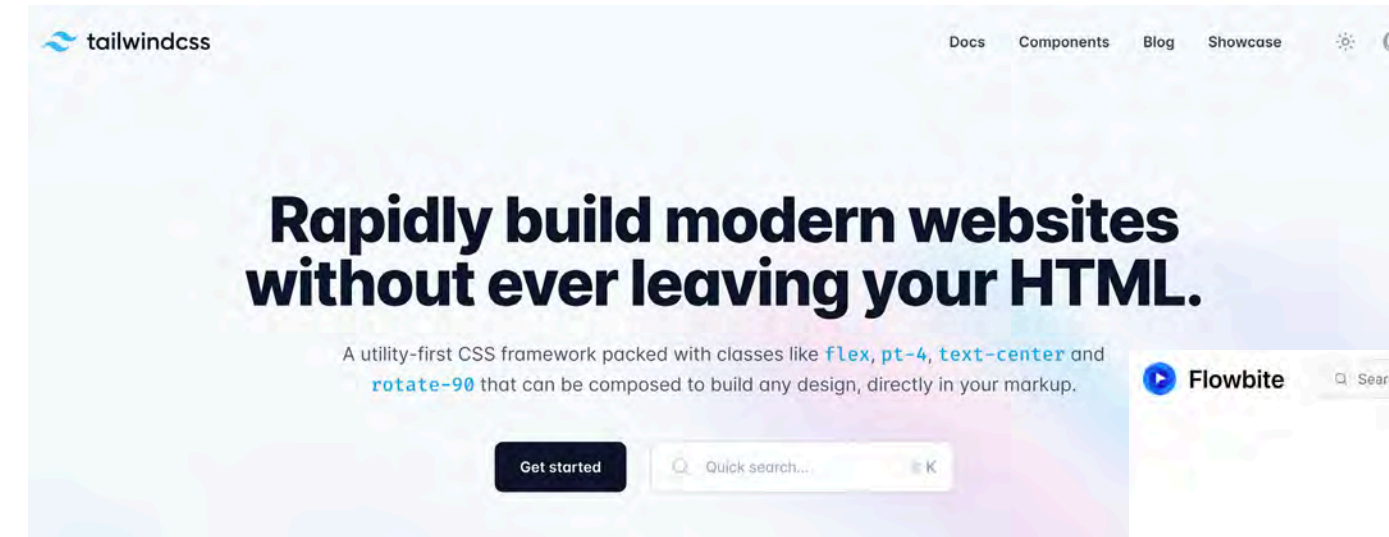
☑ 화면 구성 - CSS 프레임워크

○ Tailwind CSS

- 유틸리티 기반 CSS 프레임워크
- Tailwind 를 이용한 Flowbite 사용

☑ 기능 구현

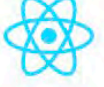
- 영화정보 API - TMDB 사용
- Data Fetching
- Custom Hooks




리액트 영화 앱 - 개발 순서 및 구현 기능

- ☑ vite 로 react 앱 스캐폴딩
- ☑ Tailwind CSS 설정
- ☑ 기능 구현
 - TMDB API 사용 - 영화 데이터 Fetching
 - 현재 상영중, 인기 상영작, 높은 평점 기능 구현
 - 영화 검색 기능 구현
- ☑ 커스텀 훅 사용으로 리팩토링
 - 재사용성과 가독성 증가

리액트 영화 앱 - 최종 구현

 리액트 무비앱

현재 상영 중 인기 상영작 높은 평점




함께 하라 새로운 시작을

트랜스포머
비스트의 서막

6월 대개봉

트랜스포머: 비스트의 서막

전 우주의 행성을 집어삼키는 절대자, '유니크론'의 부하 '스커지'는 '테러콘'들을 이끌고 지구에 당도한다. 그에 맞서기 위해 지구에 정체를 숨기고 있던 트랜스포머 '오투보' 군단이 모습을 드러내고 또 다른 트랜스포머




세상을 구할 기사가 깨어난다

세인트 세이아
더 비기닝

5월 메가박스 단독개봉

세인트 세이아: 더 비기닝

강력한 힘을 가진 전쟁과 지혜의 신 '아테나'의 환생! 세상을 구할 수호 기사가 깨어난다! 어린 시절 누나 '패트리샤'가 납치된 후 홀로 자란 '세이아' 슬럼가 지하 격투장에서 벌어지는 격투로 크크이 사아가더 쥔 위기의 사



SOUND OF
FREEDOM

사운드 오브 프리덤

설명 없음

감사합니다.

김 순 곤

soongon@hucloud.co.kr