

# 김현수\_리액트\_설명문서

## Node.js® 다운로드

Node.js® 

v22.19.0 (LTS) ▾

를

Windows ▾

환경에서

Docker ▾

방식으로

npm ▾

를(을) 사용해 설치하세요.

정보

 Want new features sooner? Get the [latest Node.js version](#) instead and try the latest improvements!

```
1 # Docker는 각 운영 체제별로 설치 지침을 제공합니다.
2 # 공식 문서는 https://docker.com/get-started/에서 확인하세요.
3
4 # Node.js Docker 이미지를 풀(Pull)하세요:
5 docker pull node:22-alpine
6
7 # Node.js 컨테이너를 생성하고 쉘 세션을 시작하세요:
8 docker run -it --rm --entrypoint sh node:22-alpine
9
10 # Verify the Node.js version:
11 node -v # Should print "v22.19.0".
12
13 npm 버전 확인:
14 npm -v # 10.9.3가 출력되어야 합니다.
```

PowerShell 

클립보드에 복사

Docker는 컨테이너화 플랫폼입니다. 문제가 발생하면 [Docker's 웹사이트](#) 를 방문하세요.

또는 

x64 ▾

 아키텍처가 실행 중인 

Windows ▾

 환경에서 미리 빌드된 Node.js®를 다운로드하세요.

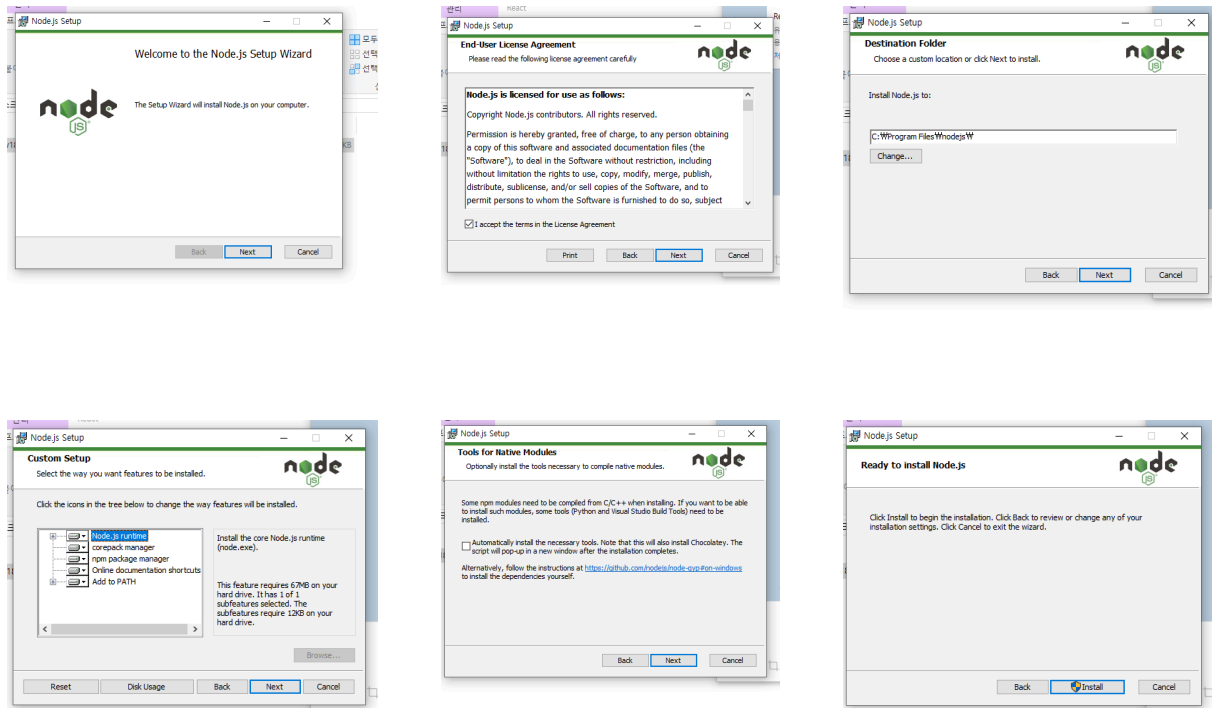
Windows 설치 프로그램 (.msi)

Standalone Binary (.zip)

공식홈페이지 : <https://nodejs.org/ko/download> (Windows 설치 프로그램.msi )

[\(기초 Node.JS설치\)](#) > [시작프로그램 \(npm 검색\)](#) > [node - v \(버전확인\)](#)

## ❖ 설치 순서 - Node.js 설치시 다른거 건들지말고 NEXT만 누를것



```
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>node -v
v18.19.0

C:\Windows\system32>
```

Node.js = 크롬 V8 자바스크립트 엔진에서 동작  
⇒ 서버사이드 런타임 환경  
⇒ (브라우저) 실행되던 자바스크립트가  
⇒ PC나 서버환경에서도 실행할 수 있게 만든 것

## NPM (Node Package Manger)

⇒ 사용하기 좋은 많은 모듈과 라이브러리를 제공

## JDK (Java Development Kit)

⇒ Java 환경에서 돌아가는 프로그램을 개발하는 데 필요한 툴을 모아놓은 소프트웨어 패키지.

# React 설치 이후 전체 흐름 정리

## 01. 설치 확인

먼저 터미널에서 버전을 먼저 확인한다.

```
node -v  
npm -v
```

버전이 나오면 정상

## 02. 작업 폴더 준비

원하는 위치에 프로젝트들을 모아둘 상위 폴더를 하나 만든다.  
예시로 C 드라이브에 React 폴더를 쓰겠다.

```
mkdir C:\React  
cd C:\React
```

## 03. 새 프로젝트 생성(CRA 기준)

Create React App으로 가장 간단히 시작한다.

```
npx create-react-app spa  
cd spa  
npm start
```

브라우저가 <http://localhost:3000> 을 열면 성공!!!

메모. Vite로 만들고 싶다면 아래 두 줄만 바꿔 쓰면 된다.

```
npm create vite@latest spa -- --template react
cd reacttest
npm install
npm run dev
```

## 04. VSCode에서 열기

VSCode를 켜고 File → Open Folder → C:\React\spa 선택.

터미널 기본 경로를 이 폴더로 고정하고 싶다면 settings.json에 다음을 추가한다.

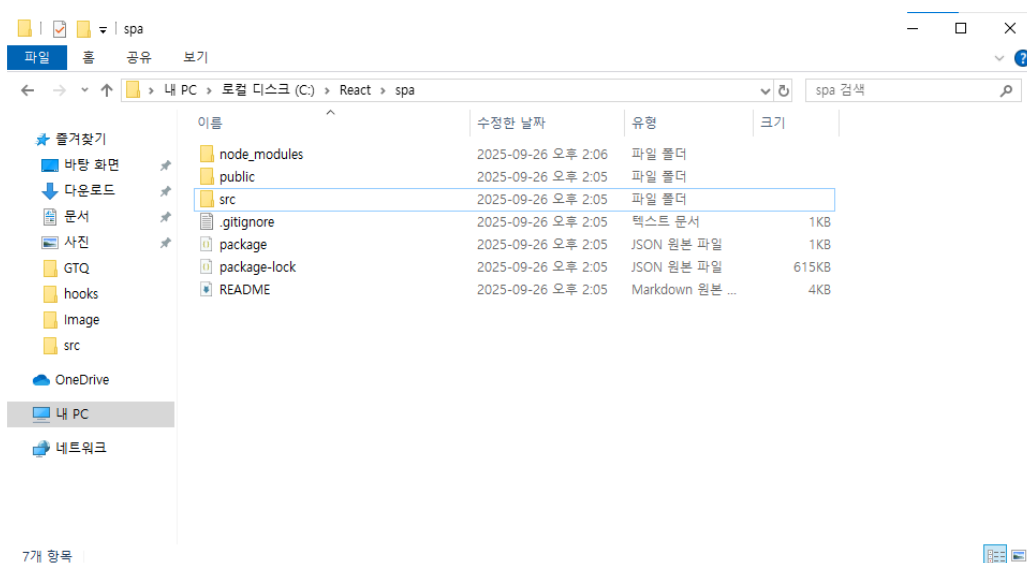
```
{
  "terminal.integrated.cwd": "C:\\React\\spa"
}
```

## 05. 폴더 구조와 핵심 파일

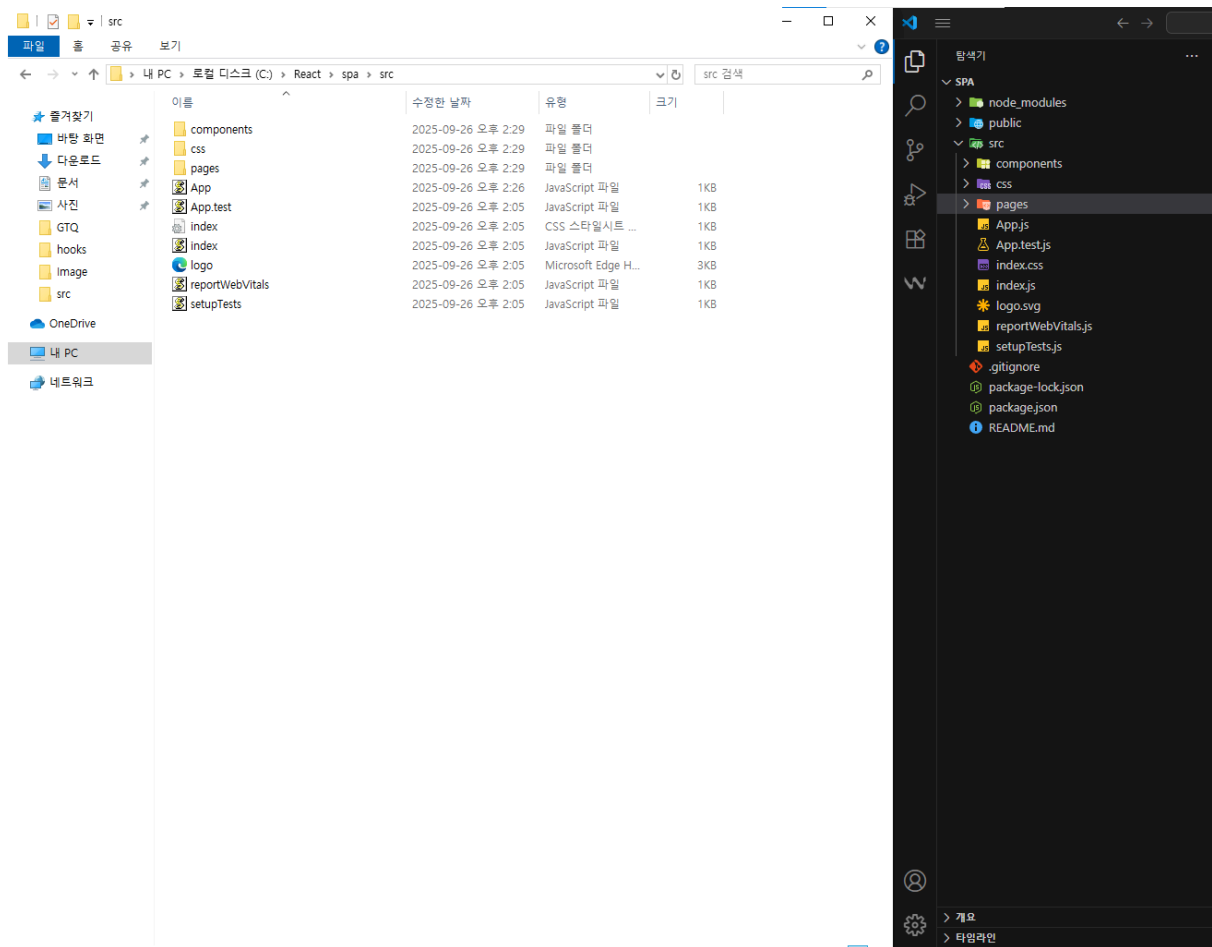
src 폴더에 실제 소스 코드를 두며,

- src/index.js 가 진입점이다. 여기서 App 컴포넌트를 root에 렌더링한다.
- src/App.js 가 화면의 시작 컴포넌트이다.
- public/index.html 의 id="root" 요소에 App이 붙는다.

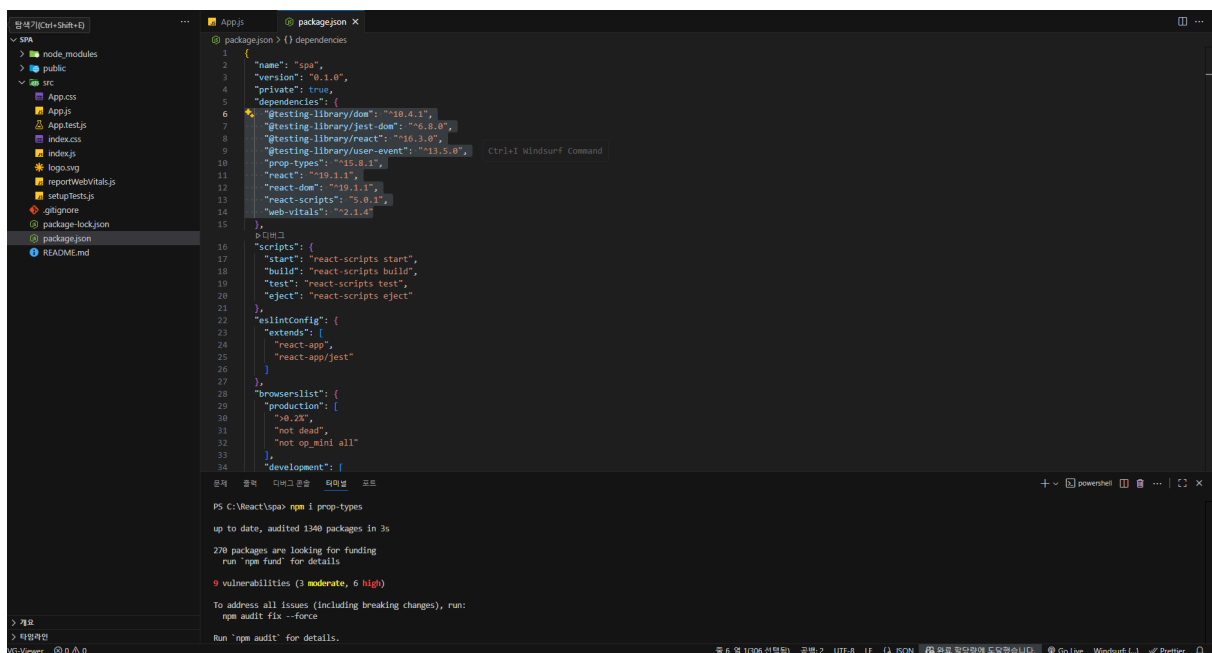
### 리액트 설치 폴더



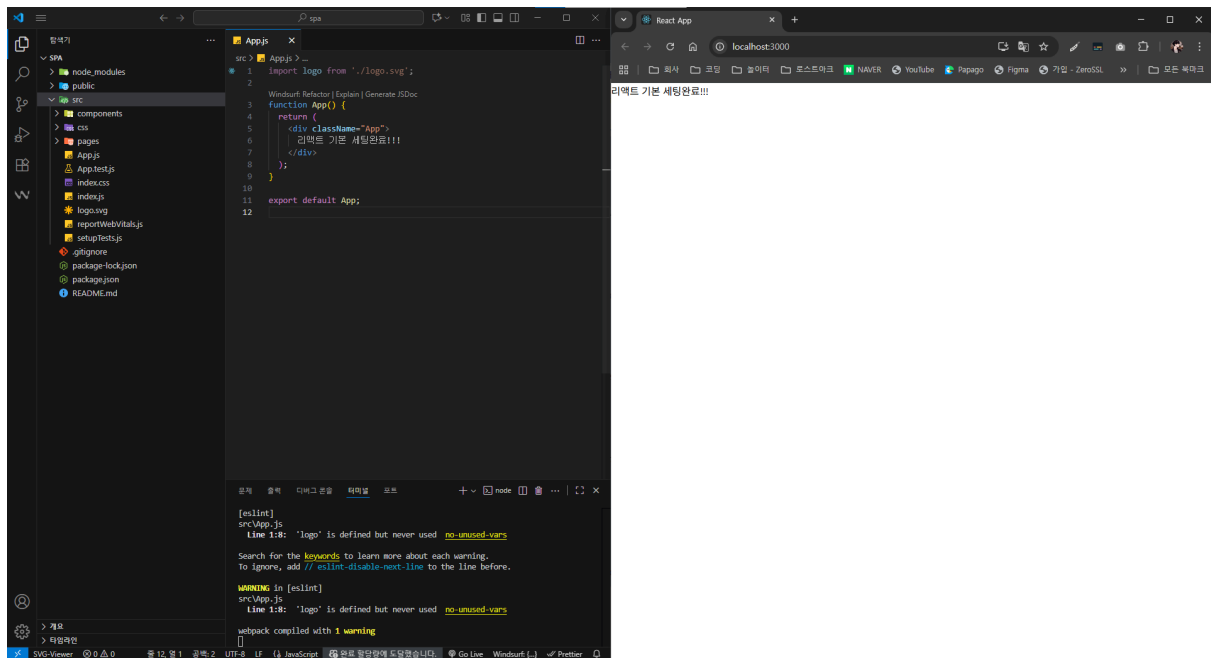
### 리액트 src 폴더



※ 기존 프로젝트를 및 확장프로그램 설치 / 버전은 한번 확인해 볼 필요가있다.



리액트 기본 셋팅 완료!!!



## 06. 컴포넌트 분리

재사용 가능한 컴포넌트는 src/components 폴더에 둔다.

```
src/
  components/
    First.js
    Second.js
    Third.js
    App.js
```

예시 코드

```
// src/components/First.js
export default function First() {
  return <h2>First Component</h2>
}
```

App에서 불러오기

```
// src/App.js
import First from './components/First'
import Second from './components/Second'
```

```
import Third from './components/Third'

export default function App() {
  return (
    <div>
      <First />
      <Second />
      <Third />
    </div>
  )
}
```

여러 학습용 App\_파일을 번갈아 실행하고 싶으면 index.js에서 렌더 대상을 바꾸거나 App.js 안에서 원하는 컴포넌트만 주석 해제하면 된다.

```
// src/index.js
import React from 'react'
import ReactDOM from 'react-dom/client'
import App_01 from './App_01_기본함수형컴포넌트'
const root = ReactDOM.createRoot(document.getElementById('root'))
root.render(<App_01 />)
```

## 07. 스타일 적용 방법 두 가지

CSS 파일을 src 안에 두고 import 하는 방식

```
/* src/css/style.css */
.font { font-weight: 700; }
.color_red { color: red; }
```

```
// src/App.js
import './css/style.css'
```

public에 둔 CSS를 link로 여는 방식

```
<!-- public/index.html →
<link rel="stylesheet" href="%PUBLIC_URL%/style.css" />
```

## 08. 자주 설치하는 기본 패키지 (\* 실무에서 거의 필수)

라우팅이 필요하다면 (라우터 개념 공부할것)

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

API 호출이 필요하다면 (axios 개념 공부할것)

```
npm install axios
```

스타일을 컴포넌트 단위로 관리하려면 (스타일 컴포넌트 공부할것)

```
npm install styled-components
```

폼 관리와 검증이 필요하다면 (hook 공부할것)

```
npm install react-hook-form yup
```

아이콘이 필요하다면 터미널에서 명령어를 치면 된다. (아이콘 불러오는법 공부할것)

```
npm install react-icons
```

## 09. npm 스크립트 이해

package.json의 scripts에 다음이 있다.

```
"start": "react-scripts start",  
"build": "react-scripts build",  
"test": "react-scripts test",  
"eject": "react-scripts eject"
```

개발은 npm start, 배포용 번들은 npm run build 이다.

## 10. 자주 만나는 오류와 빠른 해결

- ENOENT Could not read package.json 이 보이면 현재 폴더에 package.json이 없다는것 package.json이 있는 프로젝트 폴더로 이동해서 실행한다.

```
cd C:\React\reacttest
npm start
```

- Missing script: start 가 보이면 package.json의 scripts에 start가 없다는것.
- CRA로 다시 만들었는지 확인하거나 scripts를 추가한다.
- react-scripts not found 가 보이면 의존성 설치가 빠졌다는 소리.

```
npm install
```

- Port 3000 is already in use 가 보이면 다른 프로세스가 3000번 포트를 쓰는 중 y를 눌러 다른 포트로 열거나 점유 프로세스를 종료한다.

여기까지가 Node.js 설치 이후에 바로 써먹는 표준 흐름이다.

- 자주 쓰는 축약어
- **rafce** → React Arrow Function Component Export (화살표 함수 컴포넌트)

```
import React from "react"

const App = () => {
  return (
    <div>App</div>
  )
}

export default App
```

- **rfce** → React Function Component Export (기본 함수 컴포넌트 예제)

```
import React from "react"

function App() {
  return (
    <div>App</div>
  )
}
```

export default App

- `rcc` → React Class Component (클래스 함수)
- `rfc` → React Function Component (기본함수)
- `rfce` → React Function Component Export (기본 컴포넌트 예제)
- `rafce` → React Arrow Function Component Export (화살표 함수 컴포넌트)
- `rafcp` → React propTypes Function Component Export (타입 함수 컴포넌트)

## React 핵심 개념 정리

### 1. 기본 개념

#### 1. 컴포넌트 (Component)

- UI를 작은 단위로 나눠서 조립
- 재사용 가능 (버튼, 카드, 네비게이션 등)

#### 2. JSX (JavaScript + XML)

- 자바스크립트 안에서 HTML처럼 UI 구조 작성
- 코드 가독성 ↑, 작성 편의성 ↑

#### 3. Props (Properties)

- 부모 → 자식 컴포넌트로 값 전달
- 읽기 전용 데이터 (수정 불가)

#### 4. State (상태)

- 컴포넌트 내부에서 관리하는 동적 데이터
- 값이 변하면 자동으로 리렌더링

#### 5. Virtual DOM (가상 DOM)

- 변경된 부분만 실제 DOM에 반영 → 성능 최적화

#### 6. 단방향 데이터 흐름 (One-Way Data Flow)

- 데이터는 부모 → 자식으로만 전달
- 예측 가능성 ↑, 관리 쉬움

## 7. Router (라우터)

- SPA(Single Page Application)에서 페이지 전환 구현
  - 대표 라이브러리: **React Router**
  - URL에 따라 다른 컴포넌트 렌더링
- 

## 2. Hooks (함수형 컴포넌트 핵심)

1. **useState** → 상태 관리
2. **useEffect** → 사이드 이펙트 처리 (데이터 fetch, 구독 등)
3. **useContext** → 전역 데이터 공유 (props drilling 방지)
4. **useReducer** → 복잡한 상태 관리 (Redux 대체 가능)
5. **useRef** → DOM 직접 제어, 값 유지
6. **useMemo** → 값 메모이제이션 → 불필요한 연산 방지
7. **useCallback** → 함수 메모이제이션 → 불필요한 재생성 방지
8. **useLayoutEffect** → DOM 업데이트 직후 동기 실행
9. **Custom Hook** → 재사용 가능한 로직 분리

### 한 줄 요약

**React** = 컴포넌트 기반 UI 라이브러리로, **JSX + Virtual DOM + 단방향 데이터 흐름**을 바탕으로 동작하며, **Props/State**로 데이터 관리, **Router**로 화면 전환, **Hooks**로 상태와 로직을 유연하게 제어한다.