리액트, 이대로 괜찮은가?

- 모던 프론트엔드 개발 시장과 관련하여 -

프론트엔드 프레임워크의 변천사

1세대: 전통적인 SSR + DOM 조작

✅ **주요 기술**
_ **JQuery** (클라이언트)
_ **PHP, JSP, ASP.NET** (서버)

✅ **특징**
_ **서버사이드 렌더링(SSR) 중심** → 서버에서 HTML을 생성하여 브라우저에 전달
_ JavaScript는 주로 **DOM** 조작과 간단한 애니메이션을 처리하는 역할
_ **AJAX** 도입으로 페이지 전체 새로고침 없이 **비동기 데이터 통신 가능**
_ 서버에서 모든 렌더링을 담당하므로, **SEO에는 유리하지만 동적 인터랙션이 제한적**

프론트엔드 프레임워크의 변천사

2세대: SPA 개념 도입 및 CSR 확대

✅ **주요 기술**
_ **Backbone, Angular1, Knockout…**

✅ **특징**
_ **SPA(Single Page Application) 개념 등장** → 페이지 전환 없이 동적 UI 갱신
_ **JSON API + 클라이언트 렌더링(CSR) 방식** 증가
_ 클라이언트 측 라우팅 도입 → 서버 요청 없이 페이지 전환 가능
_ 그러나 초기 로딩 속도 저하, 복잡한 데이터 흐름 등 과도기적 문제 발생

프론트엔드 프레임워크의 변천사

3세대: 컴포넌트 기반 개발 & 가상 DOM

✅ **주요 기술**
_ **React, Vue.js, Angular 2+**

✅ **특징**
_ **컴포넌트 기반 개발 확립** → UI를 재사용 가능한 단위로 구조화
_ **가상 DOM 도입**으로 UI 렌더링 성능 최적화
_ 단방향 데이터 흐름(Flux 패턴, Vue의 반응형 시스템) 적용
_ 이 시기를 기점으로 **"프론트엔드 개발"이라는 개념**이 정착
_ 단순한 UI 개발을 넘어 **아키텍처**와 **상태 관리**가 중요한 요소로 부각

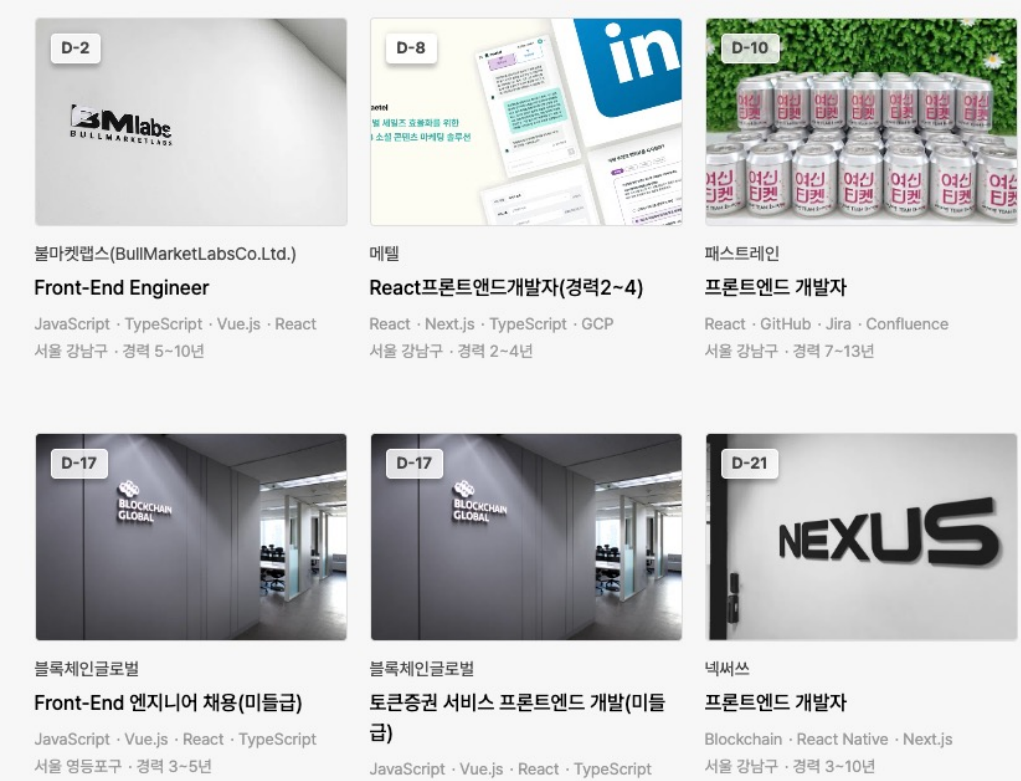
프론트엔드 프레임워크의 변천사

4세대: SSR + CSR 혼합 & 풀스택 프레임워크 (현재)

✅ **주요 기술**
_ **Next.js, Remix, Svelte, SolidJS…**

✅ **특징**
_ **SSR + CSR혼합** → 서버-클라이언트 경계를 흐리는 방식
_ **프론트엔드가 백엔드 역할까지** 수행 가능 (API 라우팅, 서버 액션 등)
_ 전통적인 SSR 방식 재조명 → SEO, 초기 로딩 속도 개선을 위한 필요성 증가
_ SPA(CSR) 방식만 고집하는 것이 **비효율적**이라는 인식 확산
_ **풀스택 프레임워크**의 등장으로 프론트엔드와 백엔드의 경계가 점점 사라지는 추세

2025년 프론트엔드 채용 공고



불마켓랩스(BullMarketLabsCo.Ltd.)
Front-End Engineer
JavaScript · TypeScript · Vue.js · React
서울 강남구 · 경력 5~10년

메텔
React프론트앤드개발자(경력2~4)
React · Next.js · TypeScript · GCP
서울 강남구 · 경력 2~4년

패스트레인
프론트엔드 개발자
React · GitHub · Jira · Confluence
서울 강남구 · 경력 7~13년

블록체인글로벌
Front-End 엔지니어 채용(미들급)
JavaScript · Vue.js · React · TypeScript
서울 영등포구 · 경력 3~5년

블록체인글로벌
토큰증권 서비스 프론트엔드 개발(미들급)
JavaScript · Vue.js · React · TypeScript

넥써쓰
프론트엔드 개발자
Blockchain · React Native · Next.js
서울 강남구 · 경력 3~10년

요구 기술 스택으로
”React”가 아닌 곳이 거의 없다!

여전히 많은 회사에서
리액트를 사용하고, 요구한다

모던 웹은 어떤 것을 충족해야 하는가?

1. 빠른 로딩 속도 & 성능 최적화

1) LCP

_ Largest Contentful Paint (최대 컨텐츠 표시 시간)
_ 웹 페이지에서 **가장 큰 컨텐츠**가 로드되는 시간 (이미지 등)
_ 사용자가 “페이지가 떴다”라고 느끼는 기준
_ 최적 목표 **2.5초** 이내

✅ How to

_ 이미지 최적화 (WebP, AVIF, SVG)
_ Lazy Loading
_ 서버 응답 속도 개선 (TTFB 최적화)

모던 웹은 어떤 것을 충족해야 하는가?

1. 빠른 로딩 속도 & 성능 최적화

2) TTFB

_ Time To First Byte (첫 바이트 응답 시간)
_ 브라우저가 서버로 요청을 보낸 후, **서버가 첫 바이트 데이터를 응답하는 시간**
_ 서버 성능과 직접적인 관련 있음
_ 최적 목표 **0.8초 이내**

✅ How to

_ **SSR**, **CDN** 활용
_ **DB Query** 최적화

모던 웹은 어떤 것을 충족해야 하는가?

1. 빠른 로딩 속도 & 성능 최적화

3) Code Splitting & Tree Shaking

_ 한 번에 모든 JS 파일을 불러오지 않고, **필요한 코드만 로드**하는 기법

_ **사용하지 않는 코드 자동으로 제거**하여 **번들 크기** 줄이는 기법

✅ How to

_ **Code Splitting**: React.lazy(), import()

_ **Tree Shaking**: ESM, Rollup, ESBuild

모던 웹은 어떤 것을 충족해야 하는가?

2. SEO & 검색 엔진 최적화

1) CSR의 SEO

_ CSR는 JS 실행 전 콘텐츠가 없다 👉 **초기 HTML이 빈 껍데기**

```
<html>
  <head><title>My Website</title></head>
  <body>
    <div id="root"></div> <!-- React가 여기에 렌더링됨 -->
    <script src="bundle.js"></script>
  </body>
</html>
```

_ 검색 엔진 크롤러(구글 봇)는 HTML을 먼저 읽어서 콘텐츠를 분석

_ 검색 엔진이 페이지 내용을 제대로 크롤링 하지 못해서 **SEO 점수가 낮아진다**!

모던 웹은 어떤 것을 충족해야 하는가?

2. SEO & 검색 엔진 최적화

2) SSR의 SEO

_ 서버에서 **완성된 HTML을 보내줌**
_ 일단 완성된 HTML을 보내주고 나중에 JS를 적용함 (Hydrate)

_ **OG / Meta 태그 / 시맨틱 태그** 등 SEO 측면에서 CSR보다 훨씬 유리해짐

모던 웹은 어떤 것을 충족해야 하는가?

3. API 최적화

1) RSC

_ React Server Components: 서버에서 실행되는 컴포넌트
_ **서버에서 데이터 페칭 & 렌더링 처리** → Next.js 써야함

2) Streaming Rendering

_ 서버가 HTML을 한꺼번에 보내지 않고, **부분적으로 전송**
_ **React Suspense** 활용 시 데이터를 기다리는 동안 필요한 UI(로딩창 등) 먼저 렌더링 가능

👉 백엔드와 데이터 페칭 방식 중요
👉 점점 복잡해지는 모던 웹에서 서버와의 긴밀한 통합 필요

모던 웹은 어떤 것을 충족해야 하는가?

4. UX/DX 최적화

1) User Experience

_ 모바일 친화적 UX 필수 (반응형, 다크 모드 등)
_ 빠른 내비게이션과 부드러운 트랜지션

2) Development Experience

_ Typescript 호환 여부
_ 복잡한 상태 관리 및 코드 최소화
_ 모바일 지원을 위한 PWA (웹 앱)
_ 코드 간결성 및 재사용 극대화

리액트가 모던 웹의 요구사항을 잘 만족시키는 부분

1️⃣ 컴포넌트 기반 아키텍처

_ 유지보수성 & 확장성이 뛰어남 → 재사용 가능한 UI 컴포넌트 설계 가능
_ 대규모 애플리케이션에서도 구조적인 코드 작성 가능

2️⃣ React Suspense & Streaming 지원

_ 데이터 페칭 최적화 → 필요한 데이터만 로드하여 성능 향상
_ 점진적 렌더링(Streaming)으로 UX 개선

3️⃣ 최대 규모의 커뮤니티 & 생태계

_ 전 세계적으로 가장 많이 사용되는 프레임워크 중 하나
_ 방대한 커뮤니티 → 문서, 튜토리얼, 라이브러리, 플러그인이 가장 많음

리액트가 모던 웹의 요구사항을 완전히 만족시키지 못하는 부분

1️⃣ **JS 번들 크기의 문제**
_ 기본 **React 프로젝트의 번들 크기가 큼** → 초기 로딩 속도 저하
_ 최적화를 해도 여전히 **최신 경량 프레임워크**(Svelte, SolidJS)보다 무거움

2️⃣ **기본적으로 CSR**
_ SEO 최적화, 초기 렌더링 성능이 부족

3️⃣ **상태 관리의 복잡성**
_ 규모가 큰 애플리케이션에서 상태 관리 어려움
_ Redux, Recoil, Zustand 같은 **외부 상태 관리 라이브러리 의존** → 추가적인 복잡성 증가

4️⃣ **런타임 오버헤드**
_ 가상 DOM에서 **Diffing 알고리즘** 실행 시 발생하는 성능 오버헤드
_ React의 UI 업데이트는 **최적화되지 않으면 불필요한 렌더링이 발생**

💬 **Fiber 아키텍처(Batch Process)는 개발자의 생산성과 성능 관리에 도움이 과연 되는 것인가?**
복잡도만 증가시킨 건 아닌가?

리액트는 모던 웹 시장에서 어떤 포지션인가?

리액트 공식 문서를 기준으로

1) CRA는 공식적으로 지원 중단했다

_ 아예 대놓고 **신규 프로젝트에 Full-Stack 프레임워크 사용하라고 권장**
_ React를 쓸 때에는 **CRA 대신 Vite**를 사용할 것을 권장

🔍 무엇을 시사하는가?

_ “이제 리액트는 프레임워크가 아니라 UI 라이브러리로써만 남을 것”
_ 리액트 자체가 프론트엔드 개발의 모든 걸 해결하는 시대는 끝났고,
적절한 생태계와 함께 사용해야 한다는 것을 인정한 셈

리액트는 모던 웹 시장에서 어떤 포지션인가?

리액트 공식 문서를 기준으로

2) “React만으로는 부족하다”는 인식 확산

_ 리액트가 처음 등장했을 땐 CSR 기반의 SPA가 대세

⚠️ **시간이 지나면서 드러난 문제들**
_ CSR의 SEO 문제
_ **첫 페이지 로딩 속도 문제**: 모든 자바스크립트를 먼저 로드해야 UI를 보여줌
_ **점점 증가하는 번들 크기**: 리액트 앱이 커질수록 로딩 성능 저하

👉 그런데 리액트 자체는 이런 문제들을 해결해주지 못했음
👉 결국 SSR/SSG 또는 하이브리드 렌더링이 필요한 시대가 됨

리액트는 모던 웹 시장에서 어떤 포지션인가?

리액트 공식 문서를 기준으로

3) 현대 프론트엔드의 구조적 문제: 복잡성 증가

_ 현재 프론트엔드 개발을 하려면 단순히 리액트만 배우면 되는 게 아님
_ 이제는 Next.js, Vite, Webpack, Babel, SWC, SSR, SSG… **온갖 개념을 알아야 함**
_ 이러한 방대한 개념들을 모두 학습하고, 적재적소에 사용할 전략까지 고려해야 함

→ **프론트엔드 개발이 점점 더 복잡해지고 어려워지고 있음**
→ 이제는 풀스택 개발이 쉬워진 만큼, **프론트엔드 개발자가 백엔드 역량까지 갖추기 쉬워짐**

💬 **“React를 배우는 것만으로는 더 이상 충분하지 않다.”**

Meta의 React 전략 변화: 프레임워크 대신 생태계

_ CSR 중심의 SPA가 유행하던 때에는
_ Meta가 **React 단독 사용**을 강조하고 **CRA**라는 공식 번들러를 제공

_ 현재 Meta는 **Full-Stack 프레임워크와의 결합을 공식적으로 인정하고** 있음

🚀 **React를 “UI 라이브러리”로 확고히 자리 잡게 하고, 풀스택 프레임워크와의 조합 강조**
❌ **자체 번들러를 유지보수하는 대신 서드파티에 의존하는 전략을 선택**

➡ 이제 React는 **개발자에게 UI 컴포넌트를 제공하는 역할**로 남고,
SSR/SSG/ISR 같은 **최적화**는 Next.js 같은 **외부 프레임워크에 맡기는 방향**으로 가고 있음.

커뮤니티의 댓글 인용

“일단 개인적으로 **현재는 리액트가 최선**이라고 생각해서 의견 드립니다.

먼저 써본 사람들의 시행착오와 라이브러리들이 갖춰진 다음에 기술 스택을 바꾸는 것이 덜 머리가 아픕니다.
시맨틱 버전으로 따지면 **최신버전 보다 하나 아래의 메이저 버전을 사용하는 셈**이죠.

요구사항과 고수준 기능은 크게 변하지 않으니 문제가 없는데,
기반기술에 대한 참고자료가 없으면 생산성이 나오지 않더군요.

B2C 서비스와 달리,
제품 사이클이 길고 유지보수 기간이 존재하는 경우에는 더더욱 최신기술을 적용하기가 어려웠습니다.

**리액트가 Next.js로 방향을 전환해서 SPA 지원을 종료하고 아키텍처 변경을 강제하는 시점에는
기술 전환을 다시 한번 고민해야 할 듯 합니다**.

Vue가 좀더 많이 보급되어 있으면 후보군에 당연히 올릴거구요. 안 쓸 이유가 없습니다. ”