React+Vite vs Next.js: 순수 SPA 시나리오

1. 상황 가정

- ✅ 백엔드가 이미 완성됨 (Python FastAPI)
- ✅ 프런트엔드는 SPA만 필요
- ✅ SEO 불필요 (B2B 금융, 로그인 후 사용)
- ✅ 프런트엔드 ↔ 백엔드 REST/GraphQL 호출만 필요

---

2. 직접 비교

React + Vite + TypeScript

장점

- ⚡ **번들링 속도**: Vite의 esbuild 사용으로 매우 빠름
- 🎯 **명확한 책임**: SPA만 집중, 백엔드와 완전 분리
- 📦 **가벼움**: 번들 크기 작음
- 🔧 **커스터마이징 자유도**: Webpack 설정 완전 제어
- 💨 **개발 속도**: HMR(Hot Module Reload) 매우 빠름
- 🏢 **팀 경험**: "순수 React"에만 집중 가능

단점

- 🏗️ 아키텍처 직접 설계 필요
- 📚 라우팅, 상태관리, 폼 등 모두 선택해서 조합
- 🚀 배포 설정 직접 구성

번들 크기 예시

```
React + Vite: ~150KB (gzip)
Next.js SPA: ~250KB (gzip)
```

Next.js (SPA 모드)

장점

- 📦 **통합 경험**: 라우팅, 상태관리 등 가이드된 구조

- 🚀 **배포 간편**: Vercel과의 완벽한 통합
- 🔍 **향후 확장성**: SSR 필요 시 쉽게 전환 가능
- 📊 **성능 분석 도구**: Vercel Analytics 자동 포함
- 🛡️ **보안**: Vercel의 자동 보안 업데이트
- 👥 **팀 온보딩**: "Next.js = 정해진 방식" → 신규 개발자 쉬움

단점

- ⚡ 개발 속도: Vite보다 약간 느림
- 📦 번들 크기: 약간 큼
- 🎯 "오버엔지니어링" 느낌: SPA만 필요한데 SSR 기능 남아있음
- 💰 비용: Vercel 프리 티어에는 제약 있음

---

3. 핵심 비교표

항목	React+Vite	Next.js
번들 크기	150KB	250KB
개발 속도	매우 빠름 ⚡⚡	빠름 ⚡
배포 설정	직접 구성	자동
라우팅	React Router (직접 선택)	내장 (파일 기반)
상태관리	Zustand/Redux (선택)	가이드됨
학습곡선	낮음 (React만)	중간 (Next.js 방식)
아키텍처 자유도	매우 높음	제한적
팀 확장성	아키텍처 설계 필요	정해진 방식
SEO 추가	매우 어려움	쉬움 (재구성)
배포 환경	어디든지 (정적 호스팅)	Vercel 최적

---

4. 결정 기준: 의사결정 트리

```
순수 SPA 필요
  |
  ├── Q1: 팀 규모?
  |   ├── 1-3명 → React+Vite ✅
  |   |   (빠른 개발, 아키텍처 자유)
  |   |
  |   ├── 3-8명 → 상황에 따라 다름 (다음 Q로)
  |   |
```

```
│   └── 8명 이상 → Next.js 추천 ✅
│     (정해진 방식으로 일관성 유지)
│
├── Q2: 향후 SEO 필요할 가능성?
│   ├── 높음 → Next.js ✅
│   │   (미리 SSR 대비)
│   │
│   └── 낮음 (순수 로그인 후 사용)
│     → 다음 Q로
│
├── Q3: 개발 속도가 최우선?
│   ├── 예 → React+Vite ✅
│   │   (Vite 빠름, 설정 최소)
│   │
│   └── 아니오 → 다음 Q로
│
├── Q4: 장기 유지보수 중요?
│   ├── 매우 → Next.js ✅
│   │   (가이드된 구조, 팀 온보딩 쉬움)
│   │
│   └── 보통 → React+Vite ✅
│     (충분한 선택)
│
└── Q5: Vercel 배포 원함?
  ├── 예 → Next.js ✅
  │
  └── 아니오 (AWS, GCP 등) → React+Vite ✅
```

---

5. 금융 포트폴리오 플랫폼 시나리오별 추천

시나리오 1: 스타트업 (초기 3개월)

팀: 당신 1명 or 2명 **기술**: Python FastAPI 백엔드 완성

추천: React+Vite ✅

이유:
- 빠른 프로토타입 필요
- 아키텍처 자유롭게 설계
- 배포는 Netlify (간단)
- 개발 속도 중요

초기 세팅 (30분)
npm create vite@latest my-app -- --template react-ts
npm install react-router-dom zustand axios
npm run dev

시나리오 2: 초기 팀 확대 (6개월, 팀 5명)

팀: 5명 (FE 3명, BE 2명) **요구**: 안정적인 구조, 신속한 개발

선택지: 둘 다 가능하지만...

React+Vite 계속 유지

장점:
- 이미 구축된 구조 유지
- 팀이 React 패턴에 익숙
- 개발 속도 유지
- 배포 파이프라인 작동 중

필요한 것:
- Monorepo (Turborepo) 도입
- 아키텍처 문서화
- ESLint, Prettier 엄격히

또는 Next.js로 이전

장점:
- 일관된 구조 강제 (팀 확대에 유리)
- 향후 확장성 높음

단점:
- 마이그레이션 비용 (1-2주)
- 개발 속도 약간 저하
- Vercel 배포 학습 필요

판단: 이 단계에서는 React+Vite 유지 추천 (아직 SEO 필요 없고, 팀이 React 능숙함)

시나리오 3: 성장 (1년, 팀 15명)

팀: 15명 (FE 8명, BE 7명) **요구**: 대규모 팀 협업, 안정성

강력 추천: Next.js로 마이그레이션 ✅

이유:
1. 팀 규모 커짐 → 정해진 방식 필요
2. 신규 입사자 많음 → 온보딩 쉬워야 함
3. 아키텍처 일관성 중요 → Next.js 방식이 강제

마이그레이션 비용:
- 2-4주 (아키텍처 완전히 바뀜)
- 하지만 새로운 팀원들 학습 시간 단축
- 장기적으로 생산성 향상

시나리오 4: 엔터프라이즈 (2년, 팀 30명+)

팀: 30명 이상 **요구**: 극도의 안정성, 규정 준수

Next.js + Monorepo (Nx) 구조 ✅

이유:
- 완전히 정해진 구조 (일관성 최고)
- 수십 팀 병렬 개발 가능
- SEO 추가 가능성 높음
- 엔터프라이즈급 보안

---

6. 실제 코드 비교

React+Vite 구조 (당신이 구축)

typescript

```
// src/main.tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import { BrowserRouter } from 'react-router-dom'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
)

// src/App.tsx
import { Routes, Route } from 'react-router-dom'
import Dashboard from './pages/Dashboard'
import Portfolio from './pages/Portfolio'

export default function App() {
  return (
    <Routes>
      <Route path="/" element={<Dashboard />} />
      <Route path="/portfolio" element={<Portfolio />} />
    </Routes>
  )
}

// vite.config.ts (완전 제어)
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  build: {
    outDir: 'dist',
    sourcemap: false,
  },
})
```

배포 (Netlify, Vercel의 정적 호스팅)

```
bash
```

```
npm run build
# dist 폴더 업로드
```

Next.js 구조 (자동화)

typescript

```typescript
// app/layout.tsx
export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="ko">
      <body>{children}</body>
    </html>
  )
}

// app/page.tsx
import Dashboard from '@/components/Dashboard'

export default function Home() {
  return <Dashboard />
}

// app/portfolio/page.tsx
import Portfolio from '@/components/Portfolio'

export default function PortfolioPage() {
  return <Portfolio />
}

// next.config.js (최소 설정)
/** @type {import('next').NextConfig} */
const nextConfig = {}
export default nextConfig
```

배포 (자동)

bash

```
npm run build
# Vercel에 git push만 하면 자동 배포
```

---

7. 결론: 당신의 금융 포트폴리오 플랫폼은?

현재 상황

- 팀 규모: 1-3명 (당신 포함)
- 백엔드: Python FastAPI (이미 완성)
- 프런트엔드: 순수 SPA만 필요
- 타임라인: 빠른 시간에 MVP 출시

최적 선택

지금 바로: React+Vite+TypeScript ✅✅✅

```
이유:
1. 빠른 개발 (Vite의 번들링 속도)
2. 아키텍처 자유 (복잡한 차트, 실시간 업데이트 최적화)
3. 배포 간단 (정적 호스팅)
4. 배우기 쉬움 (순수 React)
5. 팀이 작음 (구조 단순해도 무관)

3개월 후 팀 확대 시:
- Monorepo로 확장 (유지)
- 또는 Next.js로 마이그레이션 (재구성)
```

단, 다음 경우면 Next.js 바로 시작

- 팀이 이미 8명 이상
- 향후 SEO가 필요할 수 있음 (기업 마케팅)
- Vercel로 배포하고 싶음
- "정해진 방식"이 편함

최종 판단

선택	최적 시점
React+Vite	초기 스타트업, MVP, 빠른 개발 필요
Next.js	팀 확대, 장기 유지보수, SEO 가능성

당신의 현재 상황: React+Vite ✅

1-2년 후 팀 성장하면: Next.js로 마이그레이션 고려