

Monorepo란 무엇인가?

1. 가장 단순한 설명

Monorepo = 한 개의 저장소에 여러 프로젝트를 담기

Traditional (Polyrepo):

github.com/company/web-app ← 별도 저장소
github.com/company/mobile-app ← 별도 저장소
github.com/company/shared-library ← 별도 저장소

Monorepo:

github.com/company/portfolio ← 한 개 저장소
├── apps/web-app
├── apps/mobile-app
└── packages/shared-library

그게 끝입니다. 한 개의 git 저장소에 여러 프로젝트를 담는 것뿐입니다.

2. 구체적 예시로 이해하기

Before (지금 당신의 상황)

포트폴리오 웹앱 프로젝트 (React+Vite)

```
portfolio-web/  
├── src/  
│   ├── api/  
│   │   └── portfolioClient.ts  
│   ├── store/  
│   │   └── portfolioStore.ts  
│   ├── pages/  
│   ├── components/  
│   └── ...  
├── package.json  
├── tsconfig.json  
└── vite.config.ts
```

GitHub:

└── github.com/mycompany/portfolio-web ← 한 개 저장소

문제점

- 모바일 앱을 추가하면?

github.com/mycompany/portfolio-web ← 웹
github.com/mycompany/portfolio-mobile ← 모바일 (새 저장소)

- 두 앱이 `portfolioClient.ts` 코드를 공유하려면?
 - npm 패키지로 배포해야 함 (복잡)
 - 또는 코드 복사-붙여넣기 (유지보수 어려움)
 - 또는 git submodule (복잡함)

After (Monorepo)

한 개의 폴더 안에 모든 프로젝트를 담기

```
portfolio-monorepo/
├── apps/
│   ├── web/                ← 웹 프로젝트
│   │   ├── src/
│   │   ├── package.json
│   │   └── vite.config.ts
│   └── mobile/             ← 모바일 프로젝트 (나중에 추가)
│       ├── src/
│       ├── package.json
│       └── expo.json
├── packages/               ← 공유 코드
│   ├── core/              ← 비즈니스 로직 공유
│   │   ├── api/
│   │   │   └── portfolioClient.ts ← 웹과 모바일이 공유
│   │   └── store/
│   │       └── portfolioStore.ts ← 웹과 모바일이 공유
│   ├── types/
│   └── package.json
├── tsconfig/              ← 타입스크립트 설정 공유
├── package.json            ← 루트 설정
├── turbo.json             ← Monorepo 도구 설정
└── .github/
    └── workflows/         ← CI/CD 설정
```

GitHub:

└─── github.com/mycompany/portfolio ← 한 개 저장소 (안에 모든 것)

장점

- 웹과 모바일이 **같은 저장소**에 있음
- `portfolioClient.ts`를 한 곳에서 관리
- 웹에서 수정 → 모바일도 자동 적용
- 하나의 npm install로 모든 의존성 설치

3. 구체적 작동 원리

코드 공유의 현실

Before (별도 저장소)

typescript

```
// 웹 프로젝트 (github.com/company/portfolio-web)
// src/api/portfolioClient.ts
export const portfolioClient = {
  optimize: async (data) => { ... }
}

// 모바일 프로젝트 (github.com/company/portfolio-mobile)
// src/api/portfolioClient.ts
// 같은 코드를 다시 작성! (복사-붙여넣기 또는 npm 패키지)
export const portfolioClient = {
  optimize: async (data) => { ... }
}
```

문제: 웹에서 버그 수정 → 모바일에도 수정해야 함

After (Monorepo)

typescript

// Monorepo 루트

portfolio-monorepo/

├── packages/core/api/portfolioClient.ts

| export const portfolioClient = { optimize: ... }

|

├── apps/web/src/hooks/useOptimize.ts

| import { portfolioClient } from '@core/api' ← 공유 코드 사용

|

├── apps/mobile/src/hooks/useOptimize.ts

| import { portfolioClient } from '@core/api' ← 같은 코드 사용

장점: 한 곳에서 관리, 웹에서 수정 → 모바일도 자동 반영

4. 실제 파일 구조 변화

Step 1: 지금 (React+Vite)

portfolio-web/

├── src/

| ├── api/

| | ├── portfolioClient.ts ← API 클라이언트

| ├── store/

| | ├── portfolioStore.ts ← 상태관리

| ├── pages/

| | ├── Dashboard.tsx

| | └── Portfolio.tsx

| ├── components/

| ├── hooks/

| └── App.tsx

├── package.json

├── vite.config.ts

└── tsconfig.json

사용하는 방식

typescript

// src/pages/Dashboard.tsx

import { portfolioClient } from '@api/portfolioClient'

export function Dashboard() {

// 사용

}

Step 2: Monorepo 리팩토링 후

```
portfolio-monorepo/
|
|— packages/
|   |— core/
|       |— api/
|           |— portfolioClient.ts    ← 공유 코드로 이동
|           |— store/
|               |— portfolioStore.ts ← 공유 코드로 이동
|               |— types/
|                   |— index.ts      ← 공유 타입으로 이동
|                   |— package.json
|                   |— tsconfig.json
|
|— apps/
|   |— web/
|       |— src/
|           |— pages/
|               |— Dashboard.tsx
|               |— Portfolio.tsx
|               |— components/
|                   |— App.tsx
|                   |— package.json
|                   |— vite.config.ts
|
|— package.json          ← 루트 설정
|— turbo.json            ← Monorepo 설정
```

사용하는 방식

```
typescript

// apps/web/src/pages/Dashboard.tsx
import { portfolioClient } from '@core/api' ← 공유 코드에서 import

export function Dashboard() {
  // 동일한 코드
}
```

Step 3: 모바일 앱 추가

```
portfolio-monorepo/
```

```

|
├── packages/
│   ├── core/
│   │   ├── api/
│   │   │   ├── portfolioClient.ts    ← 웹과 모바일이 공유
│   │   │   └── store/
│   │   │       ├── portfolioStore.ts ← 웹과 모바일이 공유
│   │   │       └── ...
│   └── ...
├── apps/
│   ├── web/                ← 웹
│   │   ├── src/pages/Dashboard.tsx
│   │   └── import { portfolioClient } from '@core/api'
│   └── mobile/             ← 모바일 (새로 추가)
│       ├── src/screens/Dashboard.tsx
│       └── import { portfolioClient } from '@core/api'
│           ← 동일한 공유 코드 사용!
└── turbo.json

```

이제 두 앱 모두 같은 API 클라이언트 사용

5. 구체적인 마이그레이션 과정

현재 → Monorepo 변환 (Step by Step)

현재 상태

```

portfolio-web/
├── src/api/portfolioClient.ts
├── src/store/portfolioStore.ts
└── src/types/index.ts

```

Step 1: Turbo 설치

```

bash

npm install turbo --save-dev

```

Step 2: 폴더 구조 생성

```

bash

```

```
# 기존 파일 백업
```

```
mv portfolio-web portfolio-web-backup
```

```
# 새 구조 생성
```

```
mkdir -p portfolio-monorepo/{packages/core,apps/web}
```

```
cd portfolio-monorepo
```

Step 3: 공유 코드 이동

```
bash
```

```
# packages/core 생성
```

```
cd packages/core
```

```
npm init -y
```

```
mkdir -p api store types
```

```
# 기존 파일 복사
```

```
cp ../../portfolio-web-backup/src/api/portfolioClient.ts ./api/
```

```
cp ../../portfolio-web-backup/src/store/portfolioStore.ts ./store/
```

```
cp ../../portfolio-web-backup/src/types/index.ts ./types/
```

Step 4: 웹 코드 복사

```
bash
```

```
cd apps/web
```

```
# 기존 프로젝트 복사
```

```
cp -r ../../portfolio-web-backup/* .
```

```
# 그리고 import 경로 변경
```

Step 5: Import 경로 변경

```
typescript
```

```
// Before
```

```
import { portfolioClient } from '@api/portfolioClient'
```

```
// After
```

```
import { portfolioClient } from '@core/api'
```

6. 왜 Monorepo를 하는가?

장점

1. 코드 공유 쉬움

typescript

// 공유 코드가 자동으로 모든 앱에서 사용 가능

```
import { portfolioClient } from '@core/api'
```

```
import { usePortfolioStore } from '@core/store'
```

2. 타입 공유

typescript

// packages/core/types/index.ts

```
export interface Portfolio {
```

```
  id: string
```

```
  assets: Asset[]
```

```
}
```

// apps/web, apps/mobile 모두에서

```
import type { Portfolio } from '@core/types'
```

3. 한 번의 수정 = 모든 앱 반영

portfolioClient.ts 수정 → git push

→ 웹 자동 업데이트

→ 모바일 자동 업데이트

4. 버전 관리 통일

전체 프로젝트가 하나의 git

↓

버전 관리, 히스토리 추적 쉬움

5. CI/CD 효율화

한 번의 배포 설정 = 모든 앱에 적용

7. Monorepo의 단점

1. 초기 설정 복잡

폴더 구조 설계 필요
Turbo/Nx 학습 필요

2. 저장소 크기 증가

모든 앱의 코드가 한 곳에
→ 저장소 크기 커짐 (대부분 무관)

3. CI/CD 복잡해질 수 있음

하지만 Turbo가 자동화해줌

8. 당신의 상황에서 Monorepo 필요한가?

필요한 경우

- ✓ 웹 + 모바일 둘 다 개발
- ✓ 코드 재사용 중요
- ✓ 여러 개발자가 협업

필요 없는 경우

- ✗ 웹만 개발
- ✗ 각 앱이 완전히 다른 코드
- ✗ 혼자 개발하고 나중에 결정

당신의 상황

현재: 모바일 불확실 → Monorepo 아직 필요 없음

2-3개월 후:

- └─ 모바일 필요 → Monorepo 전환 (1주)
- └─ 모바일 불필요 → 그냥 웹만 진행

9. Monorepo 전환의 실제 과정 (당신의 경우)

Month 0-2: React+Vite로 개발

```
portfolio-web/  
├── src/  
│   ├── api/portfolioClient.ts  
│   ├── store/portfolioStore.ts  
│   └── ...  
├── package.json  
└── vite.config.ts
```

Month 2-3: 모바일 필요 판단

모바일이 필요하다면 → 다음 스텝으로

Month 3: Monorepo로 전환 (1주 소요)

하는 일

1. 폴더 구조 재조직 (30분)

```
portfolio-web/  
→  
portfolio-monorepo/  
├── apps/web/  
└── packages/core/
```

2. 공유 코드 이동 (1시간)

```
src/api → packages/core/api  
src/store → packages/core/store
```

3. Import 경로 변경 (2시간)

```
'@/api' → '@core/api'  
모든 파일에서 검색-바꾸기
```

4. 테스트 및 디버깅 (하루)

모든 기능 다시 테스트

5. Git 저장소 재구성 (1시간)

history 이전 또는 새로 시작

결과

portfolio-monorepo/

|— apps/web/ ← 기존 웹 프로젝트
|— packages/core/ ← 공유 코드
|— turbo.json ← Monorepo 설정

이제 모바일 추가 가능:

apps/mobile/

10. 구체적인 예시: 코드 수정 시나리오

Scenario: portfolioClient의 버그 수정

Before (별도 저장소)

1. github.com/company/portfolio-web
 - |— src/api/portfolioClient.ts 버그 수정
 - |— git push
2. github.com/company/portfolio-mobile
 - |— 따로 같은 버그를 수정해야 함
 - |— git push

❌ 두 저장소를 각각 관리해야 함

After (Monorepo)

1. portfolio-monorepo
 - |— packages/core/api/portfolioClient.ts 버그 수정
 - |— git push
2. 자동으로:
 - |— apps/web 재배포
 - |— apps/mobile 재배포
 - |— 둘 다 반영됨

✅ 한 곳에서 관리, 자동으로 모두 반영

최종 정리

Monorepo = 뭔가?

한 개의 git 저장소에

여러 프로젝트를 담기

실제 모습

Before:

github.com/company/web

github.com/company/mobile

After:

github.com/company/portfolio

└── apps/web

└── apps/mobile

└── packages/shared

당신의 경우

지금: 필요 없음 (웹만 개발)

2-3개월 후:

└── 모바일 필요 → Monorepo 전환 (1주)

└── 모바일 불필요 → 그대로 진행

└── 팀 확대 → 필요시 전환

전환 비용

직접 하면: 1주 (자동화 도구 사용하면 더 빠름)

외부 도움: 2-3일