금융권 채널 프론트엔드 플랫폼 구축을 위한 차세대 전략

MFE(Micro Frontend)와 DDD(Domain-Driven Design)를 통한 비즈니스 민첩성 및 확장성 확보

Next-Generation Financial Platform Strategy

왜 변화가 필요한가: 모놀리식의 한계와 리스크

The Challenge (Monolithic Rigidity)

금융 앱의 슈퍼앱(Super App)화로 인한 코드 복잡도 급증.

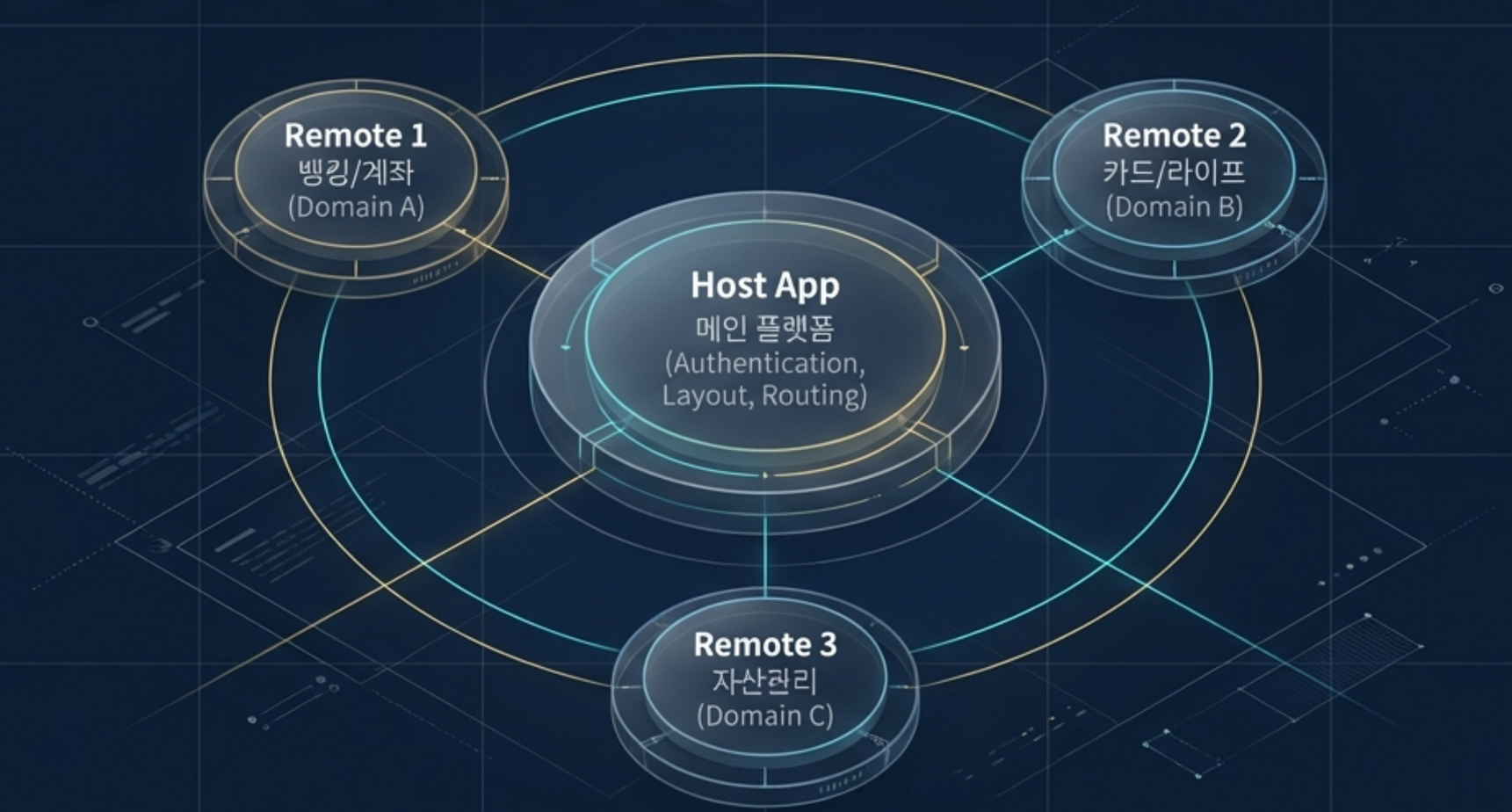
- **제한된 확장성:** 기능 하나를 수정해도 전체 애플리케이션 재배포 필요.
- **단일 실패 지점(SPOF):** '대출' 서비스의 오류가 '이체' 서비스 중단으로 이어질 위험.
- **기술 스택 경직:** 새로운 핀테크 기술 도입 시 레거시 전체를 들어내야 함.

The Solution (MFE Stability)

업무 도메인별 독립성을 보장하는 구조적 혁신.

- **장애 격리:** 서비스별 장애가 전체로 전파되지 않음 (금융권 핵심 요건).
- **독립 배포:** 타 업무 팀의 일정과 무관하게 즉시 배포 가능.

전략적 아키텍처: MFE와 DDD의 결합



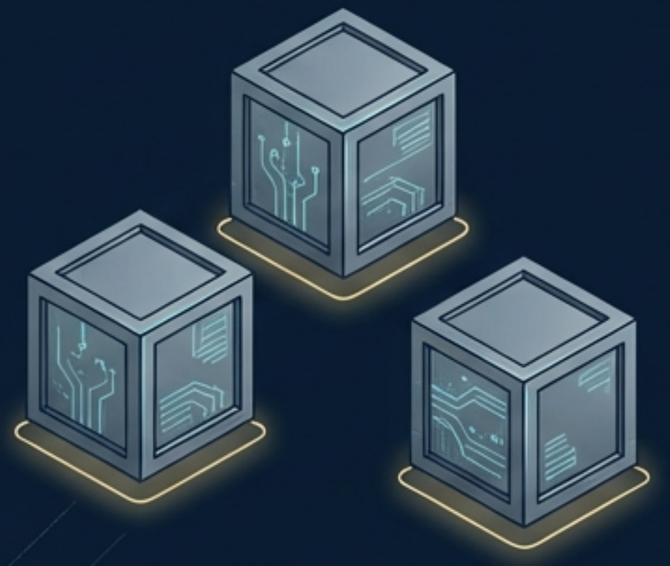
DDD(도메인 주도 설계): 비즈니스 조직(뱅킹, 카드, 보험)과 기술 모듈을 1:1로 매핑하여 책임 소재 명확화.

독립적 확장: 트래픽이 몰리는 특정 도메인(에: 공모주 청약)만 선별적으로 스케일링 가능.

레포지토리 운영 전략: 격리(Isolation) vs 통합(Integration)

금융 프로젝트의 특성(보안, 협업 규모)에 따라 레포지토리 전략이 결정됩니다.

Option A: Multirepo (다중 저장소)



각 서비스가 물리적으로 완전히
분리된 Git 저장소를 가짐.
'완벽한 독립성.'

**우리 프로젝트에
최적화된 모델은 무엇인가?**

Option B: Monorepo (단일 저장소)

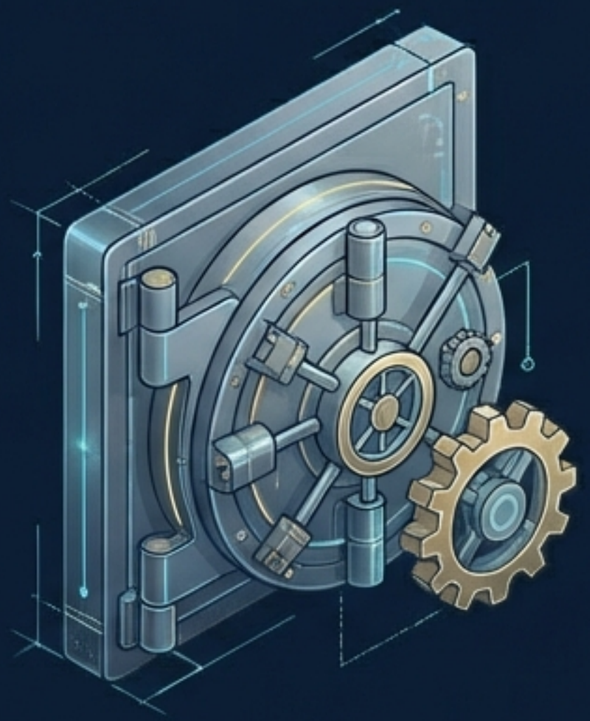
하나의 저장소에서 여러 프로젝트와
공통 라이브러리를 통합 관리.
'운영 효율성.'

심층 분석: 멀티레포 (Multirepo) 전략

Strict Independence & Security Boundaries

Pros

- **보안/권한 격리:** 외주 인력이나 특정 팀에게 특정 저장소 접근 권한만 부여 가능 (금융권 망분리/보안 요건 유리).
- **배포 파이프라인 분리:** CI/CD가 물리적으로 분리되어 있어 타 팀의 빌드 실패 영향 전무.
- **기술 스택 자율성:** A팀(React), B팀(Vue) 등 완전히 다른 기술셋 허용 가능.



Cons

- **공통 모듈 관리 비용:** 공통 라이브러리 수정 시 'npm publish' 및 각 레포별 버전 업데이트 필요.
- **통합 테스트 난이도:** 여러 레포에 걸친 변경 사항을 한 번에 테스트하기 어려움.
- **코드 중복:** 저장소 간 유틸리티 함수나 설정 파일 중복 발생 가능성.

심층 분석: 모노레포 (Monorepo) 전략

Unified Efficiency & Developer Velocity

Pros

- **단일 버전의 진실** (Single Source of Truth): 'pnpm workspaces'를 통해 모든 의존성을 중앙 관리, 유령 의존성(Phantom Dependencies) 방지.
- **원자적 커밋** (Atomic Commits): API 변경과 이를 사용하는 UI 수정이 하나의 커밋으로 처리됨 (리팩토링 용이).
- **개발 생산성:** 로컬에서 전체 앱을 한 번에 구동 및 디버깅 가능. 심볼릭 링크(Symlink)로 공통 라이브러리 수정 즉시 반영.

Cons

- **빌드 복잡성:** 'Turborepo'나 'Nx' 같은 정교한 빌드 도구 및 캐싱 전략 필수.
- **권한 관리의 한계:** 저장소 전체에 대한 접근 권한이 필요할 수 있어, 디렉토리별 정교한 권한 제어(CODEOWNERS) 필요.

금융권 프로젝트를 위한 선택 기준 및 제언

Security/Outsourcing

외부 파트너 협업 비중이 높음
→ Multirepo 유리

Efficiency/Standardization

전사 표준 UI/UX 및
빠른 기능 개발 중요
→ Monorepo 유리

CORE PLATFORM RECOMMENDATION

Strategic Verdict

모노레포(Monorepo) 방식 권장.

Why Text

MFE 환경에서 공통 라이브러리('@rm/mf-shared-library')의 잦은 변경과 통합 비용을 최소화하기 위함.

핵심 뱅킹 플랫폼은 모노레포로 구축하되, 완전히 득립적인 제휴 서비스는 멀티레포로 분리하는 하이브리드 전략 검토 가능.

전사 표준화 전략: 공통 라이브러리 (@rm/mf-shared-library)

Enterprise Standardization Strategy: Shared Library

Workflow

모노레포 환경:
'workspace:*' 프로토콜을 통한 즉시 참조 (빌드 시간 단축).

멀티레포 환경:
Private NPM Registry를 통한 버전 관리 및 배포.

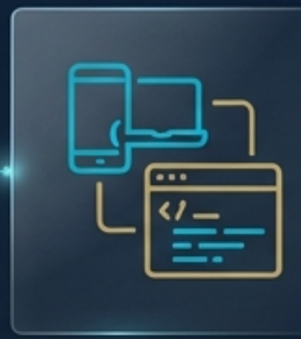
품질 및 거버넌스 체계: 일관성 유지 방안

Linting Strategy

- ESLint Flat Config 도입.
- eslint-config-prettier로 포맷팅 충돌 방지.

Strict Rules

- 'no-explicit-any' (타입 안정성).
- 'react-hooks/exhaustive-deps' (버그 방지).



IDE Synchronization

- '.vscode/settings.json' 공유.
- 모든 개발자가 동일한 저장/포맷팅 환경 사용.

Package Manager

- 'pnpm' 도입.
- 디스크 효율성 증대 및 유령 의존성 문제 원천 차단.

런타임 통합 기술: Vite & Module Federation

Build Engine: Vite
(ESBuild 기반 초고속 빌드)

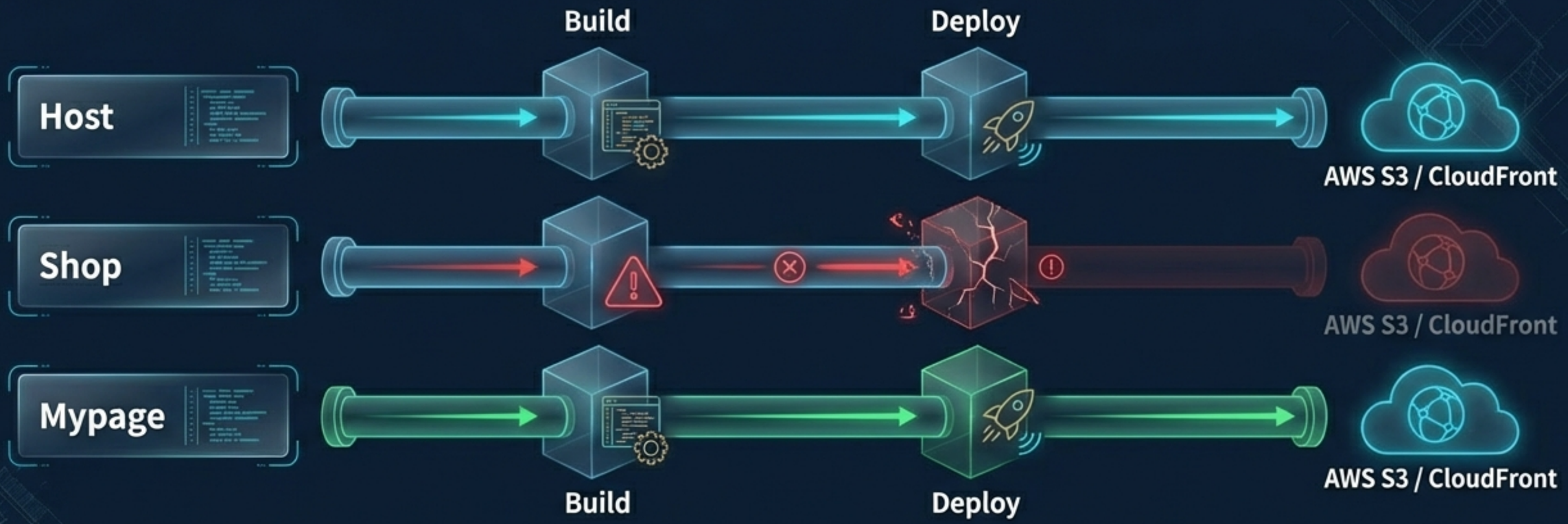
Runtime Loading: Module Federation
(원격 모듈 동적 로딩)

No Page Reload:
페이지 이동 없이 부드러운 앱 경험(SPA) 제공.

Dependency Sharing:
React, ReactDOM 등 핵심 라이브러리는 한 번만 로드하여 성능 최적화 (Singleton).

Version Independence:
서로 다른 배포 주기를 가진 앱들이 런타임에서 하나의 서비스로 결합.

배포 및 DevOps 전략: 독립 배포와 안정성



Build Once, Deploy Anywhere:
각 마이크로 앱은 독립적인 빌드 파이프라인 보유.

Fault Isolation Strategy:
'쇼핑' 앱 배포 중 에러가 발생해도 '뱅킹' 앱은 정상 가동.

Versioning:
시맨틱 버저닝(Semantic Versioning)을 통한 명확한 변경 이력 관리.

개발 생산성 및 DX (Developer Experience) 혁신

Unified Workspace:
'pnpm workspaces'를 통해 루트에서 'pnpm install' 한 번으로 모든 환경 세팅 완료.

Type Safety:
'tsconfig.base.json' 상속을 통한 전사적인 타입스크립트 정책 통일.

Proxy Integration:
로컬 개발시 'vite.config.ts'의 Proxy 설정을 통해 실서버 API와 유연하게 연동.

Automation:
Git Hook 및 CI 자동화를 통한 코드 품질 검사(Lint/Test) 강제화.

결론: 미래를 위한 플랫폼 로드맵

유연하고, 안전하며, 효율적인 금융 서비스 구축을 위한 최적의 전략입니다.

Standardization:
공통 라이브러리 기반의 UI/UX 일관성 및 품질 보장.

Operation:
모노레포를 통한 관리 효율성 및 협업 속도 증대.

Architecture:
MFE + DDD로 비즈니스 변화에 유연한 구조 확립.