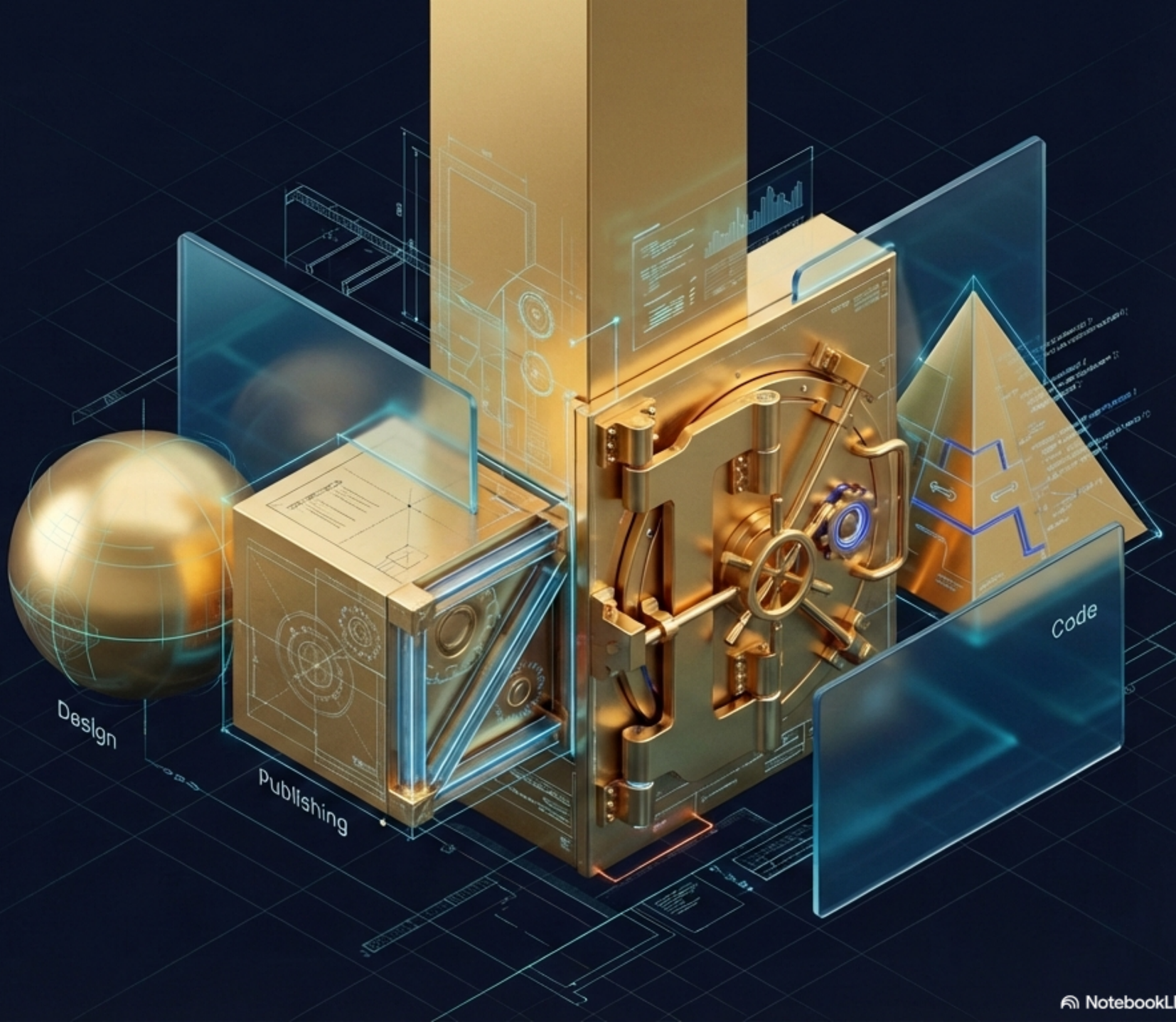
# 프론트엔드 플랫폼 표준화 전략

디자인, 퍼블리싱, 프론트엔드 개발의 유연한 아키텍처와 효율적 협업을위한 제안



# 추진 배경 및 핵심 목표

## 거대하고 복잡한 금융 시스템의 한계를 넘어, 민첩성과 안정성을 동시에 확보해야 합니다.

AS-IS

Monolith

TO-BE

Agility & Order

### Monolith의 한계

거대한 단일 시스템으로 인한 배포 지연 및 유지보수의 어려움 해소.

### Agility (민첩성)

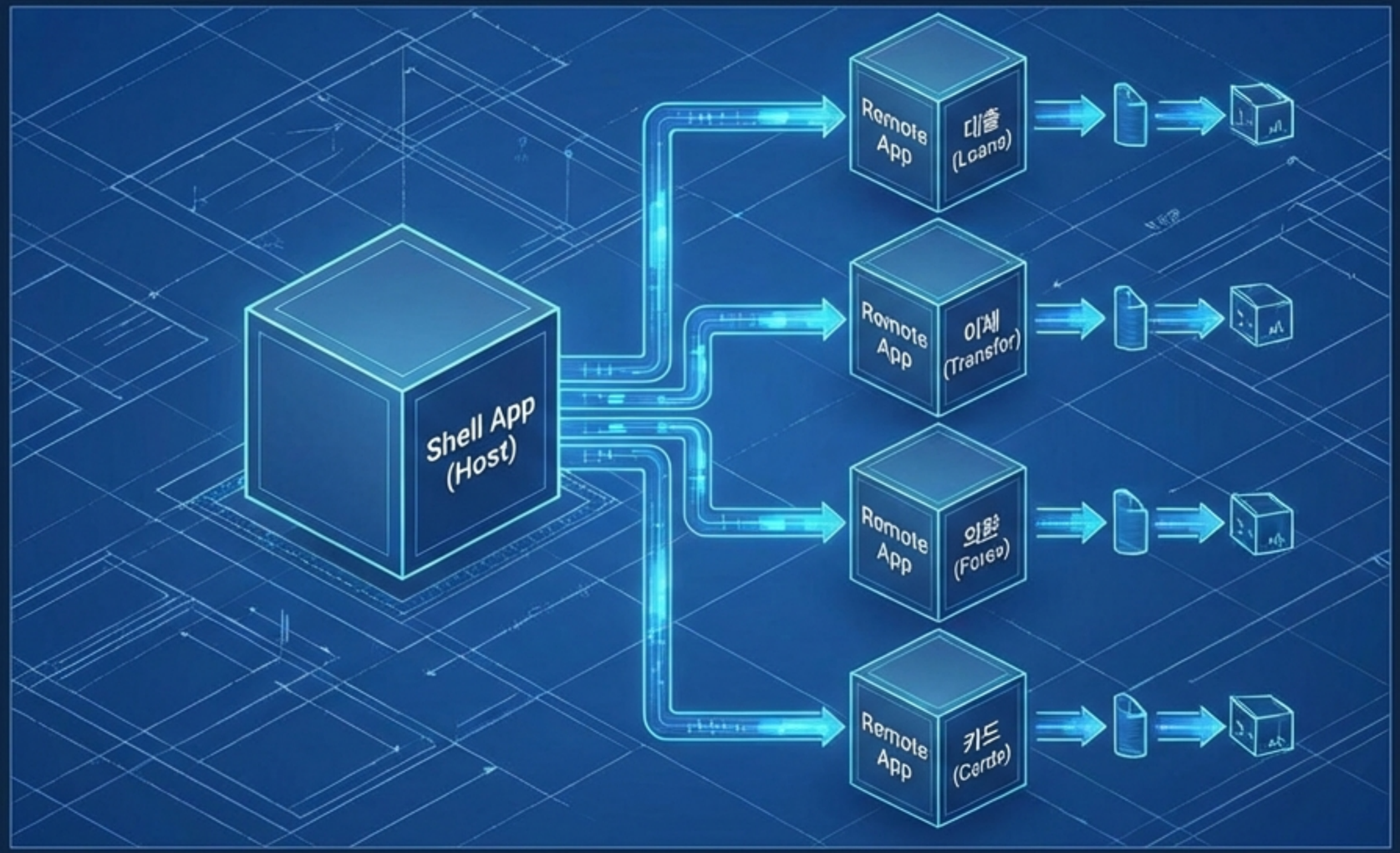
업무(도메인)별 독립적인 배포와 빌드가 가능한 환경 구축.

### Consistency (일관성)

디자인부터 개발까지 끊김 없는(Seamless) 워크플로우 정립.

# 아키텍처 전략: 마이크로 프론트엔드 (MFE)

## 독립적으로 배포 가능한 수직적 분할 구조




**Technical Strategy:**

- 런타임 통합: Vite + Module Federation을 활용한 런타임 런타임 통합 방식 적용.

- 이점: 뱅킹, 대출, 외환 등 업무별 개별 빌드/배포 가능. 장애 전파 차단 (Risk Isolation).

# Based on Slide C

## Monorepo (모노레포)


- 특징: 단일 저장소에서 모든 프로젝트 관리.
- 추천 상황: 공통 UI 라이브러리 의존도가 높고, 팀 간 협업이 긴밀하며, 일관된 개발 환경이 중요할 때.

## Multirepo (멀티레포)


- 특징: 서비스별 독립적인 저장소 운영.
- 추천 상황: 조직 규모가 매우 크고(100명+), 업무 단위가 완전히 분리되어 있으며, 배포 주기가 상이할 때.

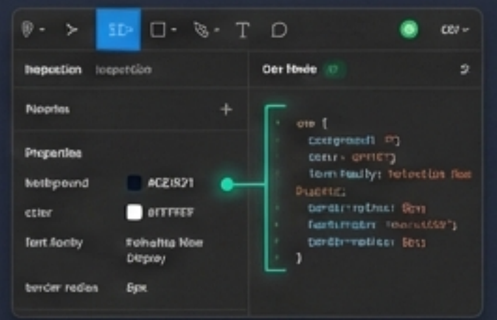
결론: 금융권의 보안 요건과 공통 모듈 관리를 고려하여 최적의 방안을 선택해야 함.

# 통합 개발 워크플로우 (Integrated Development Workflow)

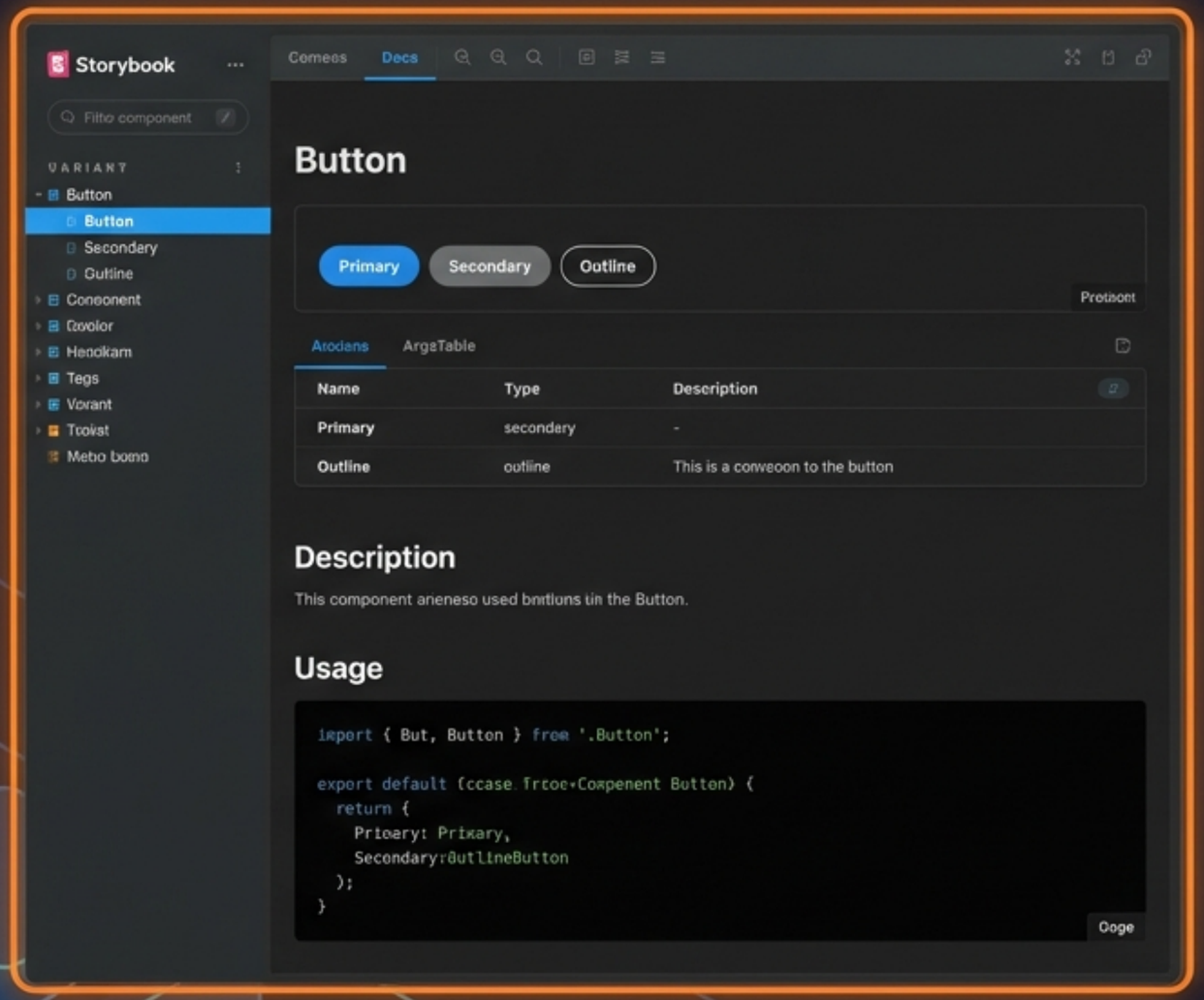
# 디자인 작업 영역 및 전략 (Design Scope & Strategy)


## Figma

- **Tool:** Figma
- **Key Activities:**
  - **아토믹 디자인:** 작은 요소(원자)를 조합하여 전체 디자인 시스템 구축.
  - **디자인 토큰:** 색상, 글꼴 등 공통 요소를 변수(Variable)로 일관되게 관리.
  - **개발자 핸드오프:** Dev Mode를 통해 정확한 디자인 사양과 코드를 전달.



# 퍼블리싱 작업 영역: Component Driven Development (CDD) with Storybook

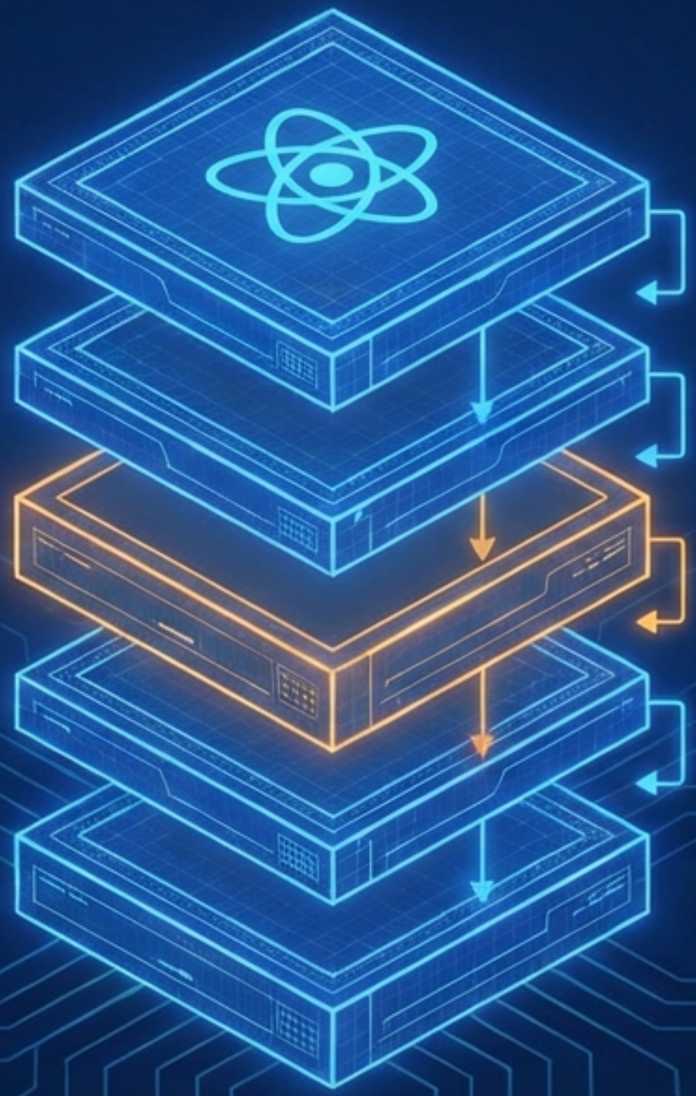



**Tool:** Storybook

**Key Activities:**

- **공통 UI 컴포넌트 개발:** 아토믹 디자인 기반의 재사용 가능한 컴포넌트 (예: 버튼, 입력 필드)를 독립적으로 구현 및 문서화.
- **개별 업무 화면 구성:** 공통 컴포넌트를 조합하여 비즈니스 로직 없이 개별 업무 화면(Page/Template)을 Storybook에서 구성 및 검증.
- **자동화된 문서화 (Autodocs):** Storybook Autodocs 및 MDX를 활용하여 '사용자 매뉴얼' 자동 생성.
- **결과물 (Deliverable):** 모든 UI 컴포넌트와 업무 화면이 정리된, 상호작용 가능한 Storybook URL 제공.

# 프론트엔드 핵심 기술 스택 (Frontend Core Tech Stack)



Core Framework

React.js (컴포넌트 기반 UI 구축)

Language

TypeScript (정적 타입 지정을 통한 코드 품질 향상)

Micro-frontend Architecture

Module Federation (마이크로 프론트엔드 구현)

Build Tool

Vite (빠른 빌드 속도 및 개발 환경)

Quality Control

ESLint + Prettier (코드 품질 및 스타일 통일)

# UI 컴포넌트 전략: Shadcn/ui & Tailwind

```
inport { Button } fron "@/components/ui/button"

<Button Variant="outline">Click He</Button>

// Copy this component code directly into your project
```

Copy & Paste

## 유연성 및 확장성 (Flexibility & Scalability)

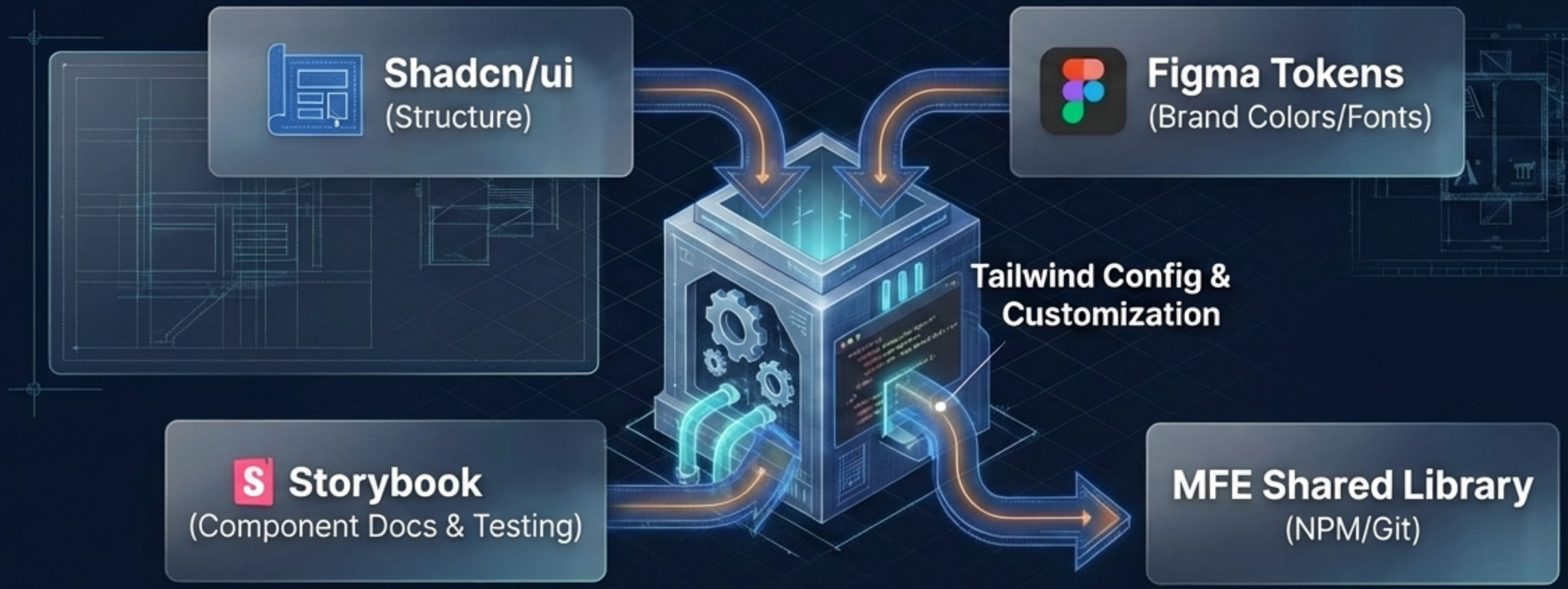

**Strategy:** Headless UI 기반의 소스 코드 소유형 라이브러리 채택.

**Why Shadcn/ui?**

- **Ownership:** 라이브러리 종속성(Dependency) 없이 코드를 직접
- **Base**: Radix UI 기반의 웹 접근성(Accessibility) 기본 준수.
- **Styling**: Tailwind CSS를 활용한 유틸리티 퍼스트 스타일링.

- Headless UI 기반으로 다양한 UI 라이브러리 (예: Bootstrap) 쉽게 통합 및 전환 가능.
- 금융권 요구사항에 맞춰 디자인 시스템 유연하게 변경 가능.

# 공통 UI 컴포넌트 표준화 방안 (Common UI Standardization)




🏗️ **기본 (Base):** Shadcn/ui를 통해 기본적인 공통 UI 골격 확보.
🖌️ **커스터마이징 (Customization):** 피그마 디자인 토큰을 Tailwind Config에 적용하여 커스텀 테마 입히기.
📦 **배포 (Distribution):** Storybook으로 검증된 컴포넌트를 사내 패키지(NPM/Git)로 배포하여 전사 MFE 공유.

# 프론트엔드 개발 및 연동 프로세스 (Frontend Dev Process)

```tsx
import { Button } from '@bank/ui'
import { useForm } from 'react-hook-form'
import { useQuery } from '@tanstack/react-query'
import { zodResolver } from '@hookform/resolvers/zod'
import { userSchema } from './validation'

const UserForm = () => {
  const { register, handleSubmit } = useForm({ resolver: zodResolver(userSchema) })
  const { data, isLoading } = useQuery({ queryKey: ['user'], queryFn: fetchUser })

  if (isLoading) return <div className="text-blue-500">Loading...</div>

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="p-4 border border-gray-300 rounded-
      <Button type="submit" variant="primary">Submit</Button>
    </form>
  )
}
```

## Role Definition:

**Input:** Storybook에 정의된 UI 컴포넌트 및 Props 명세.

**Task:**

- React Hook Form, Zod 등을 활용한 폼 데이터 처리.
- Tanstack Query 등을 이용한 서버 통신 및 상태 관리.
- **핵심**: 공유 라이브러리의 UI 컴포넌트를 임포트(Import)하여 조립(Composition).

# MFE 통합 메커니즘: Vite Plugin Federation


**Key Tech:**

- Shared Dependencies:
  공통 모듈은 쉐어링하여 번들 사이즈 최적화.

- Runtime Loading:
  사용자가 메뉴 클릭 시 해당 자바스크립트 번들을 동적으로 로딩.

# R&R: 역할 및 책임의 명확화 (Roles & Responsibilities)

## Designer

- UI/UX 설계
- 디자인 시스템(Token) 정의
- Figma Dev Mode 가이드

## Publisher

- Storybook 기반 UI 컴포넌트 개발
- 반응형 처리 및 웹 접근성
- 문서화(MDX) 작성
- shadcn/ui, TailwindCSS 활용

## Frontend Dev

- 비즈니스 로직 및 API 연동
- 상태 관리 (State Management)
- Storybook 컴포넌트 참조 및 연동
- MFE 연동 및 배포 파이프라인

# 단계별 구축 로드맵 (Implementation Roadmap)

NEW

(was Phase 3)

(was Phase 4)

## Phase 1: Foundation

01

모노레포/멀티레포 결정 및 초기 환경 설정 (Vite, ESLint).

## Phase 2: Design System

02

Figma Token 정의 및 Shadcn/ui 기반 공통 라이브러리 구축 (Storybook)

## Phase 3: UI Commonization & MFE Setup


03

UI 공통화, 마이크로 프론트엔드 구축: 공통 UI 컴포넌트 개발 및 MFE 아키텍처 기반의 통합 환경 구축.

## Phase 4: Pilot Development

04

핵심 업무(Domain) 1개를 선정하여 MFE 구조로 선행 개발.

## Phase 5: Expansion

05

전사 업무로 확대 및 배포 파이프라인 자동화.

# 결론: 지속 가능한 금융 플랫폼을 위하여

## Conclusion: For a Sustainable Financial Platform

- **Standardization:** 디자인 시스템 기반의 UI 표준화 달성.
- **Efficiency:** 퍼블리싱과 개발의 명확한 분업 및 MFE를 통한 배포 속도 향상.
- **Future-Proof:** 기술 종속성을 최소화(Shadcn)하고 유연성(MFE)을 극대화한 아키텍처.