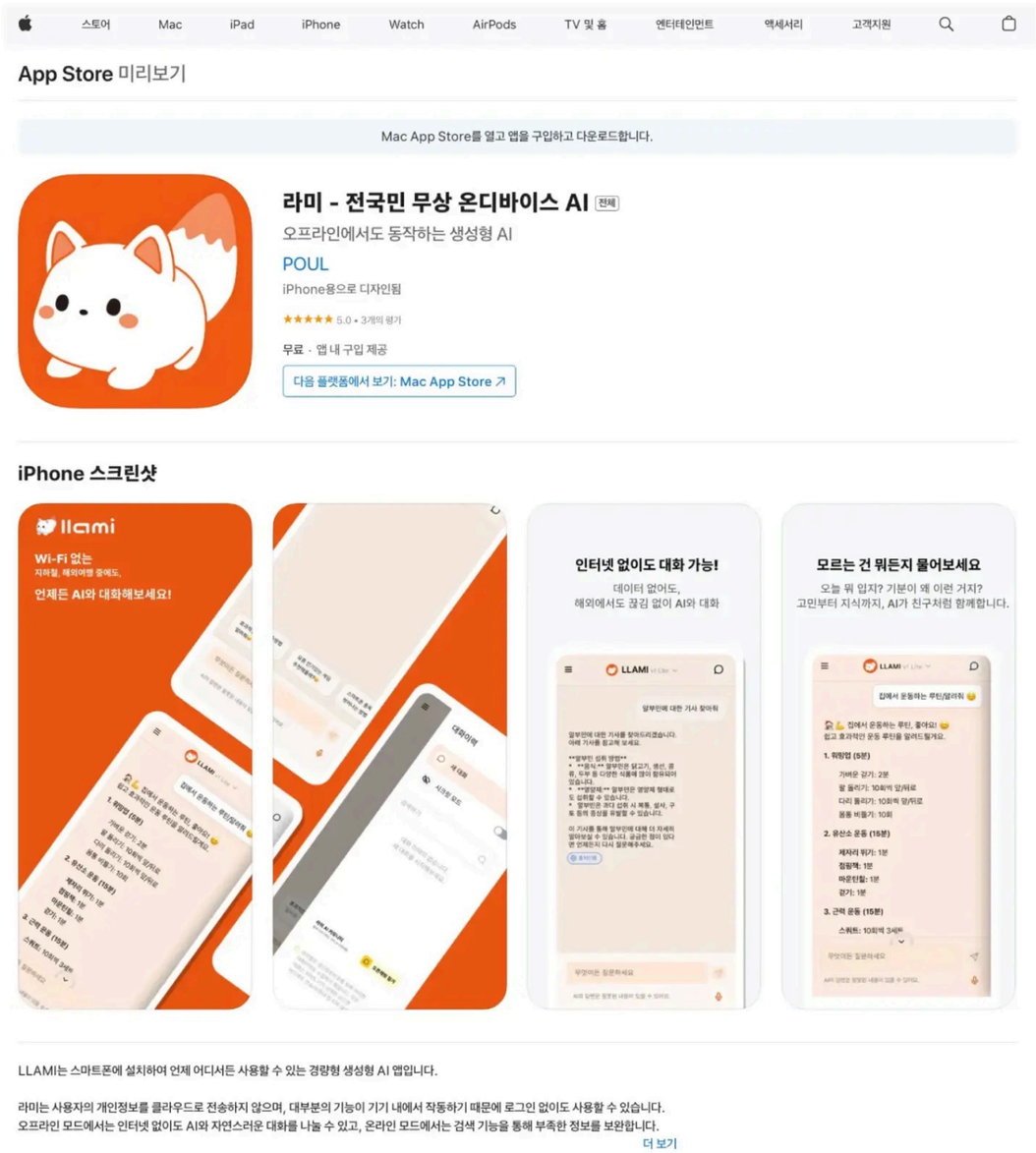
🎯 라미앱 개발 배경

비즈니스 문제 정의



진행기간

2025.05.04 ~ 2025.06.17 출시

맡은 역할

CTO 및 리드 개발자

프로젝트 기여도

30% (네이티브 앱 아키텍팅 및 PoC 구현)

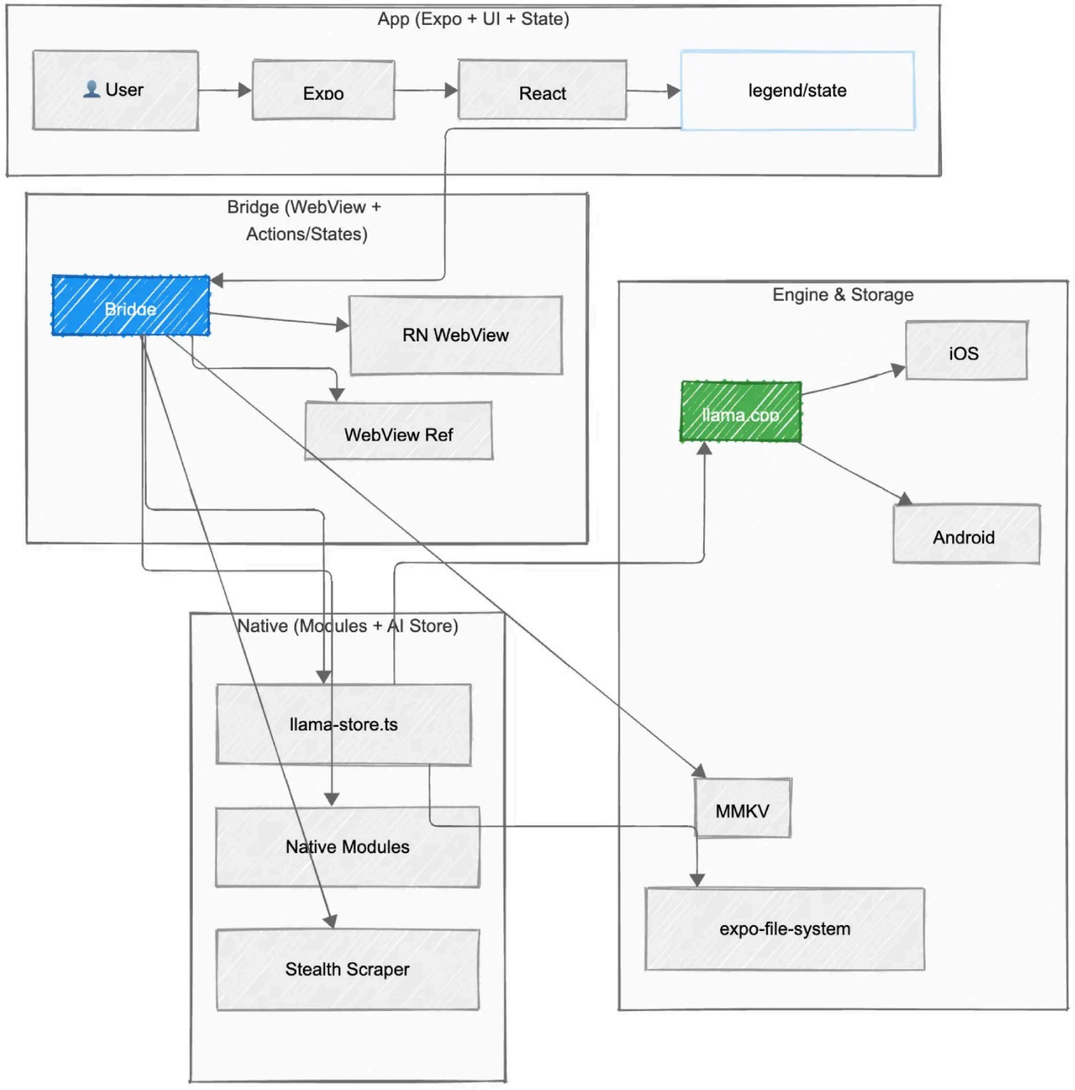
비즈니스 문제

- 결제 페이지에서 **95% 이탈률**
- 고객 인터뷰: "무료 AI 서비스가 많아 결제 망설임"
- CAC가 투자자 요구 대비 **5배 초과**
- 기존 AI 서비스 고정비 절감 방안은 대규모 서비스에만 특화

"AI 서비스 고정비를 획기적으로 줄일 새로운 방안이 요구됨"

🏗️ 라미앱 설계

요구사항과 해결 방안



요구사항 1: 고정비 제로화

→ 온디바이스 AI 앱 설계로 서버 비용 완전 제거

요구사항 2: 웹 팀으로 앱 개발

→ Expo 웹뷰 + 핵심 모듈만 React Native/C++ 하이브리드 설계

요구사항 3: 자체 AI 모델 운용

→ Gemma3 SLM 파인튜닝 후 GGUF 모델로 배포

💻 라미앱 구현

기술적 도전과 해결

핵심 구현 사항

1. 상태 브릿지 시스템

- Observable State 기반 Expo ↔ React Native 상태 공유
- 점진적 마이그레이션 가능한 아키텍처

2. AI 모델 업데이트 시스템

- Cloudflare R2 기반 GGUF 모델 배포
- 앱 재설치 없이 모델 업데이트 가능

3. 크로스 플랫폼 AI 가속

- llama.cpp 통한 iOS Metal GPU 가속
- Android CPU 최적화 추론

4. 개발 생산성 최적화

- 웹 개발자가 네이티브 지식 없이 개발
- 타입 안전성 보장된 브릿지 패턴

기술적 의사결정

- React 컴포넌트로 빠른 MVP 구현 → 점진적 React Native 전환
- 브릿지 패턴 추상화로 중복 코드 60% 감소
- EAS Build 자동화로 빌드 시간 50% 단축
- Observable 기반 상태 관리로 타입 안전성 확보

🚀 라미앱 결과

비즈니스 임팩트

0원

AI 고정비 달성

100% 온디바이스 기술

500%

고객 참여도 향상

유료 AI 대비 5배

2.6배

개발 생산성 향상

MVP 6주 → 2.3주

프로젝트를 통해 배운 점

비즈니스 임팩트

- 기술적 제약을 비즈니스 기회로 전환
- 온디바이스 AI로 고정비 제로화 달성
- Cloudflare R2로 배포 비용 최적화

기술적 의사결정

- 빠른 개발 vs 좋은 설계의 균형점 찾기
- 점진적 마이그레이션 전략의 중요성
- 팀 역량에 맞는 기술 스택 선택

🎯 라미 로보틱스 - 4족보행 로봇

DeepRobotics 로봇개 원격 제어 및 자율 주행 플랫폼

프로젝트 기간

2025.07.16 ~ 2025.09.01 (중단)

역할

CTO 및 리드 개발자

프로젝트 기여도

이하민 50%: 딥로봇틱스 Lite3 로봇개 원격 제어 및 자율 주행 시스템 개발. WebRTC 기반 실시간 영상 스트리밍과 WebSocket 통신을 통한 로봇 제어, SLAM 기반 맵핑 및 경로 계획 기능 구현.

핵심 기술 과제

로봇 웹 통신 프로토콜 부재
딥로봇틱스 Lite3는 UDP 바이너리 프로토콜 사용, 공식 웹 SDK 없음

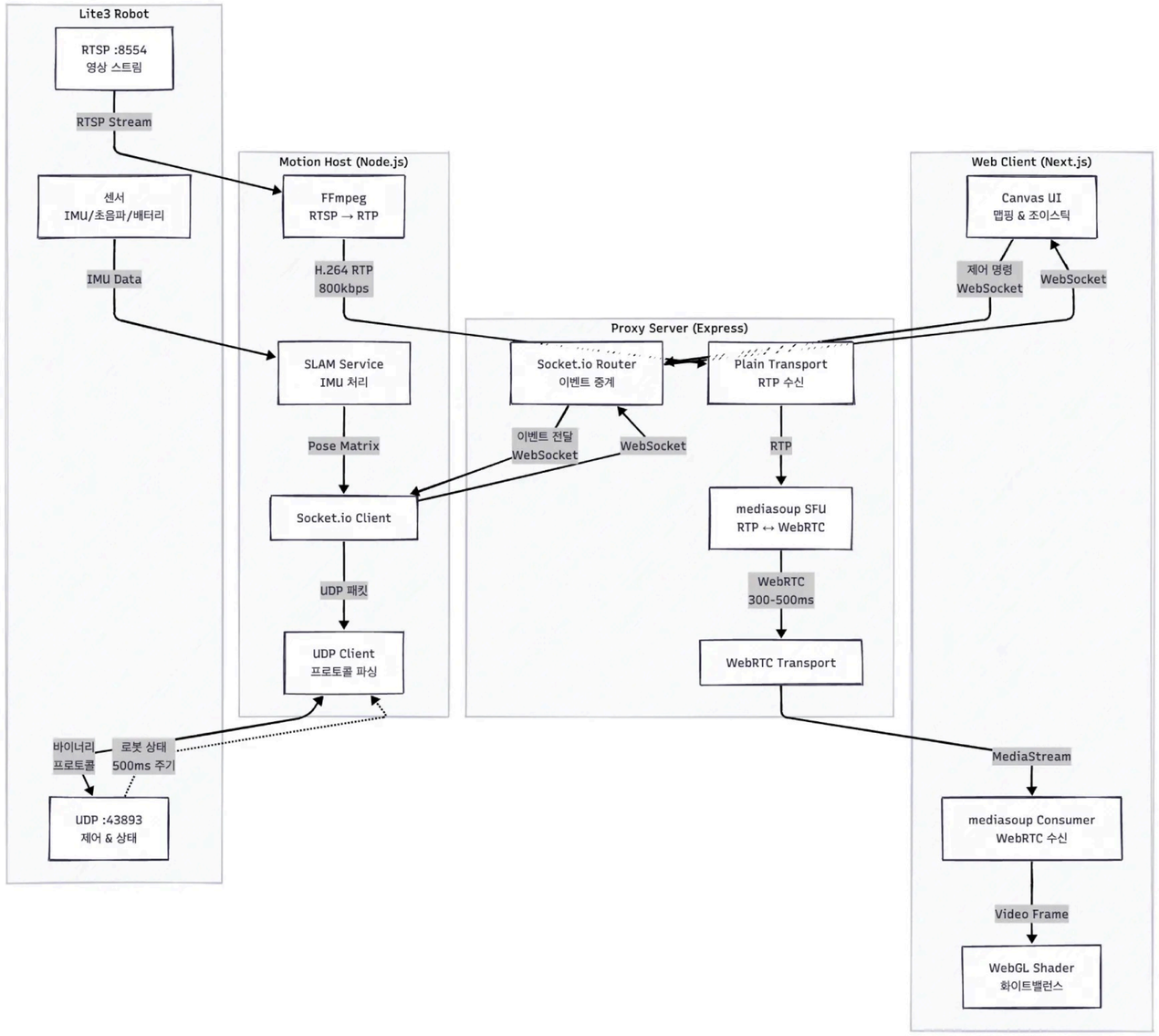
실시간 영상 전송 지연
RTSP 스트림을 웹에서 직접 재생 불가, HLS는 3~5초 지연

복잡한 환경에서의 경로 계획
장애물 회피, 작업 영역 제약, 선호 경로 등 다층 제약 조건 처리

영상 품질 및 성능 문제
로봇 카메라의 불안정한 화이트밸런스, CPU 기반 영상 처리는 30fps에서 프레임 드롭 발생

라미 로보틱스 설계

핵심 설계 결정



1. UDP 프로토콜 직접 구현

- Node.js dgram 모듈로 UDP 클라이언트 구현
- 프로토콜 헤더 구조 분석 (12바이트 헤더 + 가변 데이터)
- 500ms 주기 Heartbeat로 연결 유지

2. mediasoup SFU 기반 WebRTC 스트리밍

- RTSP → RTP → WebRTC 변환 파이프라인
- FFmpeg ultrafast preset으로 인코딩 지연 최소화
- NACK/PLI 피드백으로 패킷 손실 복구

3. A* 알고리즘 + 다층 비용 맵

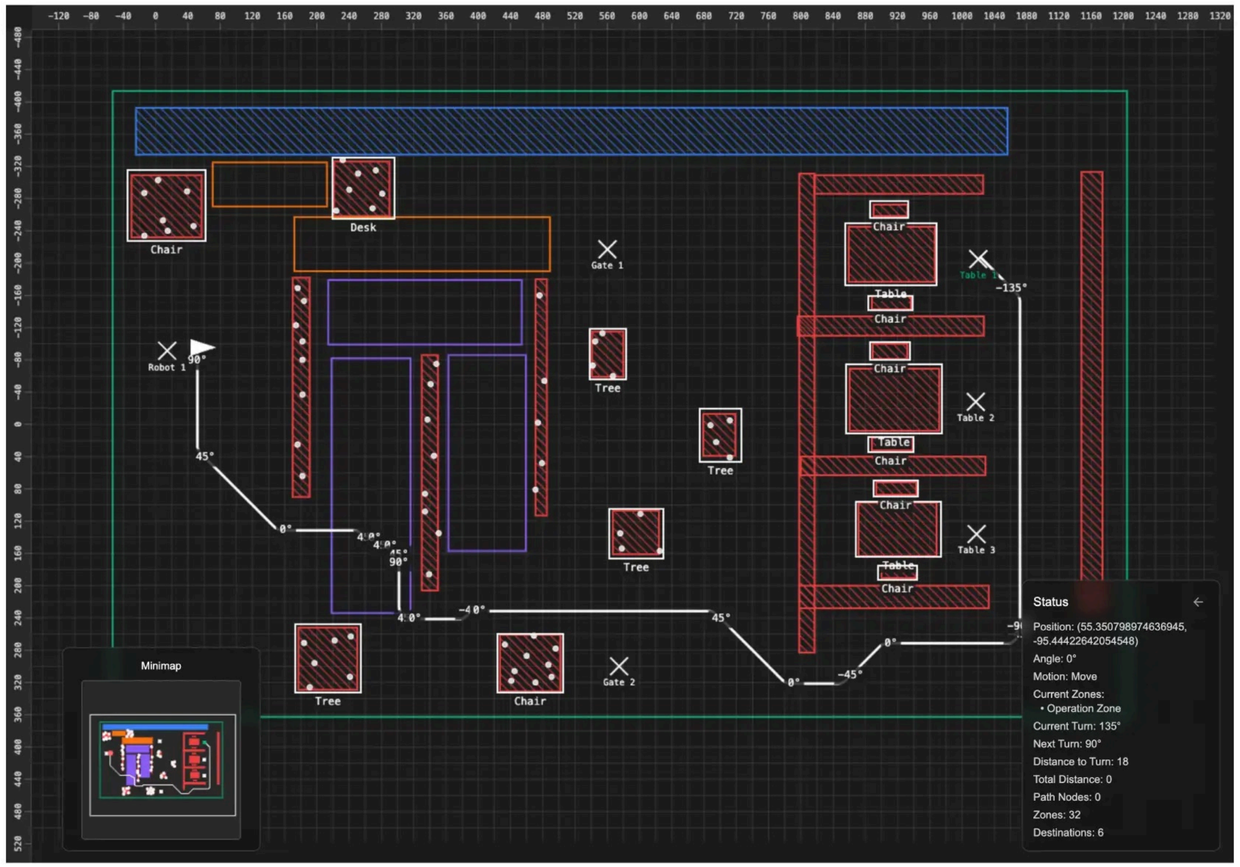
- 5가지 Zone 타입: Obstacle, Avoidance, Preferred, Operation, Slow
- 비용 맵 기반 경로 계획: Obstacle=∞, Avoidance=10, Preferred=0.5

4. WebGL 셰이더 기반 화이트밸런스

- GPU 가속으로 실시간 30fps 처리
- Gray World Assumption 기반 자동 색온도 보정

라미 로보틱스 구현

기술적 도전과 해결



1. UDP 로봇 제어 프로토콜

상황: 공식 SDK 없이 바이너리 프로토콜 직접 구현 필요

행동: Wireshark로 패킷 분석, 200+ 바이트 로봇 상태 데이터 파싱

영향: 실시간 양방향 통신 달성 (500ms 주기)

2. WebRTC 저지연 영상 스트리밍

상황: RTSP를 웹에서 재생 불가, HLS는 3~5초 지연

행동: FFmpeg로 RTSP → H.264 RTP 트랜스코딩, mediasoup로 WebRTC 전송

영향: 지연 시간 3~5초 → 300~500ms (90% 감소)

3. Canvas 기반 A* 경로 계획

상황: 복잡한 환경에서 자율 주행 필요

행동: 다층 비용 맵 생성, 8방향 이웃 탐색 A* 알고리즘 구현

영향: 4000×4000 가상 월드에서 경로 계획 평균 45ms

4. WebGL 화이트밸런스 & SLAM 시각화

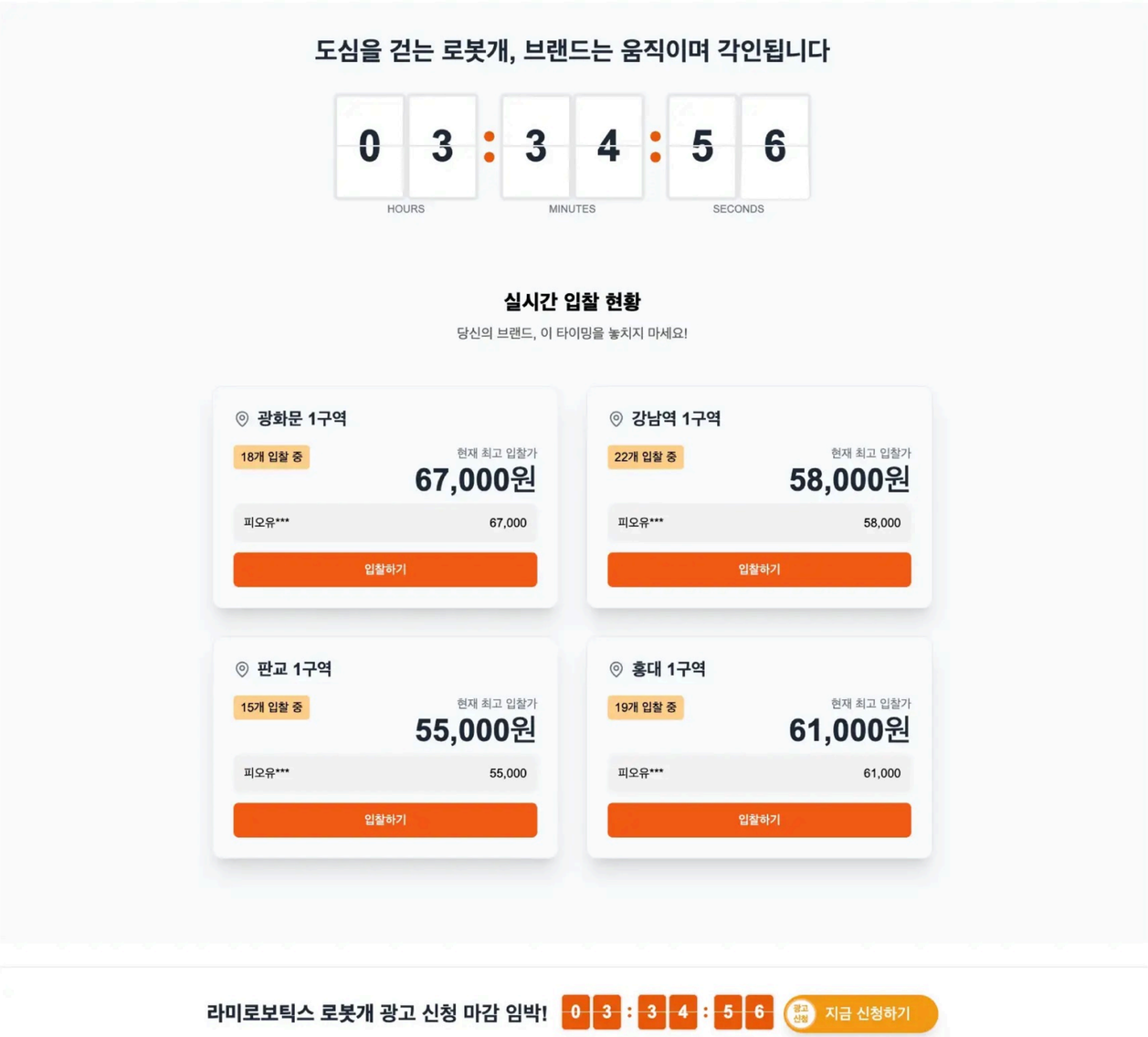
상황: CPU 기반 영상 처리는 프레임 드롭 발생

행동: WebGL Fragment Shader로 GPU 가속 구현

영향: 1920×1080 해상도 30fps 실시간 처리, CPU 사용률 5% 이하

🚀 라미 로보틱스 결과

비즈니스 임팩트 및 학습



비즈니스 임팩트

90% 영상 지연 감소 3~5초 → 300~500ms	**45ms** 경로 계획 속도 실시간 대응 가능
30fps 영상 처리 성능 CPU 5% 이하	**500ms** 제어 응답성 실시간 양방향 통신

기술적 의사결정

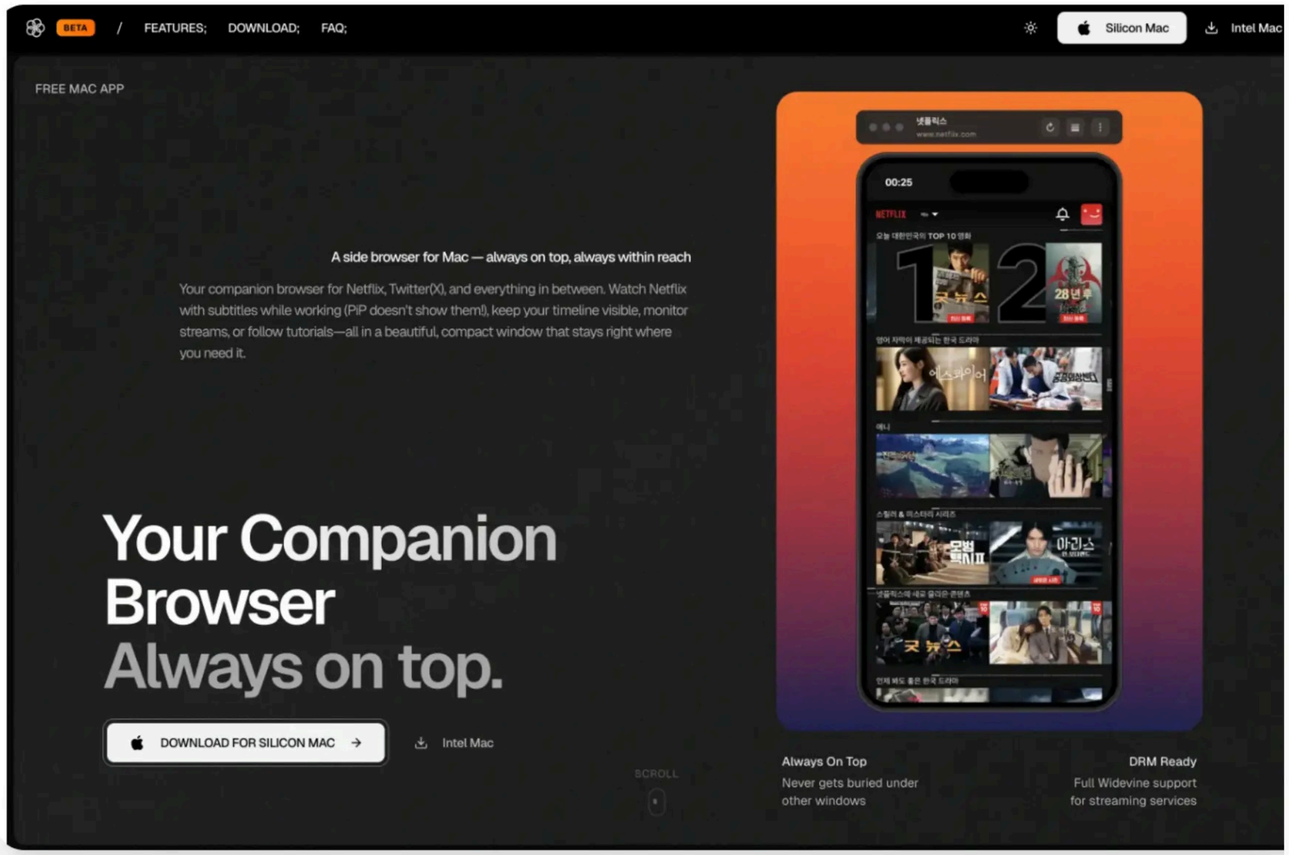
- Turborepo 모노레포로 apps 분리 (web, motion-host, proxy)
- LegendState Observable 패턴으로 반응형 상태 관리
- WebGL GPU 가속으로 CPU 부하 최소화
- mediasoup SFU로 다중 클라이언트 확장 가능

프로젝트를 통해 배운 점

- 저수준 UDP 프로토콜 구현 및 바이너리 데이터 파싱 경험
- WebRTC SFU 아키텍처 설계 및 실시간 스트리밍 최적화
- 경로 계획 알고리즘의 실제 로봇 제어 적용

🎯 aka-browser: 미니 브라우저 개발

프로젝트 개요 및 배경



진행기간

2025.10월 2주 진행 (오픈소스)

맡은 역할

개인 프로젝트 (설계, 개발, 배포)

기술 스택

Electron (Castlabs), React 18, TypeScript, Vite

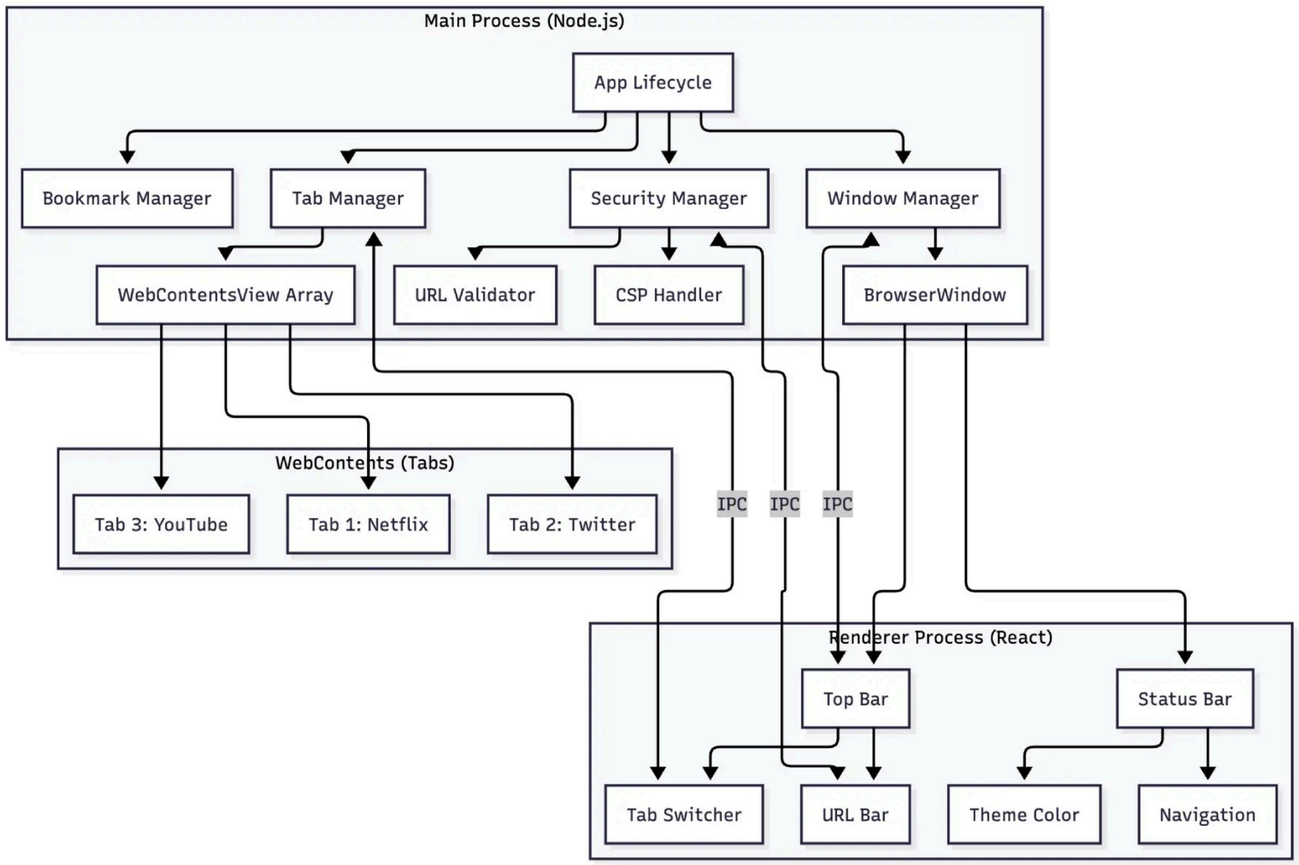
사용자 페인 포인트

- 웹 / Expo 개발 시 iOS 시뮬레이터는 노트북에서 항상켜기 너무 무거움
- Netflix를 PiP 모드로 볼 경우 자막이 PiP에 표시되지 않음
- DRM 콘텐츠 재생이 가능하며 항상 위에 떠있는 미니 브라우저 부재

"작업하면서 항상 위에 떠있는, 가벼운 세컨드 브라우저가 필요하다"

🏗️ aka-browser 설계

시스템 아키텍처



1. 관심사 분리

- TabManager: 탭 생명주기 관리
- WindowManager: 윈도우 레이아웃 관리
- SecurityManager: URL 검증 + CSP
- BookmarkManager: 즐겨찾기 CRUD

2. 이벤트 기반 통신

- IPC 채널: 메인 ↔ 렌더러 프로세스 분리
- 타입 안전성: TypeScript 인터페이스 정의
- 보안 검증: 모든 IPC 요청에서 sender 검증

3. 메모리 최적화

- 비활성 탭: removeChildView()로 렌더링 중단
- LRU 캐시: 테마 컬러 100개 도메인 저장
- 탭 프리뷰: 800x1200 해상도 제한

💻 aka-browser 구현

핵심 구현 사항

핵심 구현 사항

Widevine DRM 통합

- Castlabs Electron + Component Updater
- EVS VMP 서명 자동화 (afterPack 훅)
- 도메인별 User Agent 전환

보안 시스템

- 프로토콜/도메인 검증 + IPC sender 검증
- CSP: 개발/프로덕션 환경 분리

멀티탭 시스템

- WebContentsView 생성 + 이벤트 등록
- 프리뷰 캡처 → 탭 교체 → 테마 적용
- 탭 전환 <100ms, LRU 캐시 최적화

추가 기능

- 북마크: JSON CRUD + IPC 동기화
- 에러 페이지: 100+ 에러 코드 매핑

기술적 도전과 해결

	문제	✓ 해결
🔴 Widevine 서명 실패	수동 VMP 서명 시 빌드마다 반복 작업 필요	electron-builder afterPack 훅으로 서명 자동화
🔴 화면 깜빡임	탭 전환 시 테마 컬러 재계산으로 UI 지연	LRU 캐시로 도메인별 테마 컬러 저장 (100개)
🔴 보안 vs 기능	DRM 재생 위해 sandbox 비활성화 필요	프로토콜/도메인/IPC/CSP 4중 검증 체계 구축

🚀 aka-browser 결과

프로젝트 성과

<100ms	~150MB	100%	4중
탭 전환	**메모리**	**빌드 자동화**	**보안 검증**
프리뷰 캐싱	3탭 기준	수동 작업 제로	프로토콜/도메인/IPC/CSP

정성적 성과

✅ 실제 서비스 운영

GitHub 오픈소스 + macOS 배포

✅ 기술 부채 관리

TypeScript 엄격 모드 + 모듈화

✅ 성능 최적화

LRU 캐시 + 렌더링 최적화

핵심 인사이트

빠른 변화 대응

Netflix UA 정책 변경 → 도메인별 UA 전환 (2시간)

기술 부채 예방

정적 HTML → React TSX 동적 생성

서비스 안정성

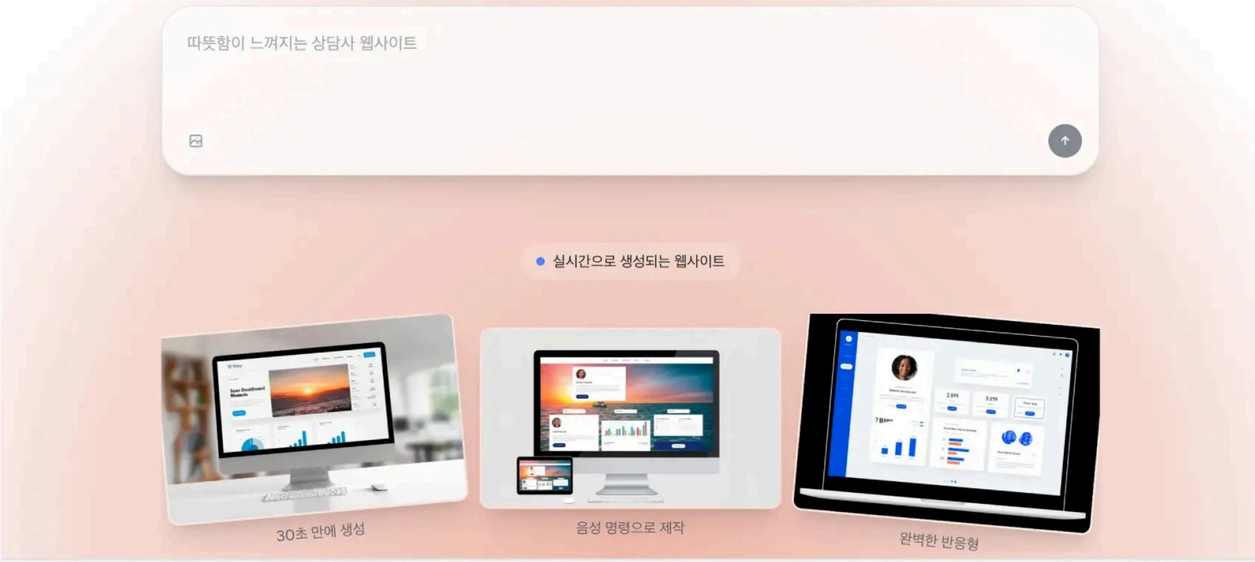
Widevine 검증 실패 → EVS 서명 자동화

보안과 기능 균형

sandbox 비활성화 → 4중 검증으로 보안 유지

🎯 큐리오시티 개발 배경

비개발자의 빠른 MVP 검증을 위한 온프레미스 AI 코딩 플랫폼



비즈니스 문제 정의

비개발자의 MVP 검증 장벽

- 외주 개발 비용: 최소 500만원 이상, 2-3개월 소요
- 노코드 툴의 한계: 커스터마이징 불가능
- 개발자 채용: 스타트업 초기 단계에서 인건비 부담

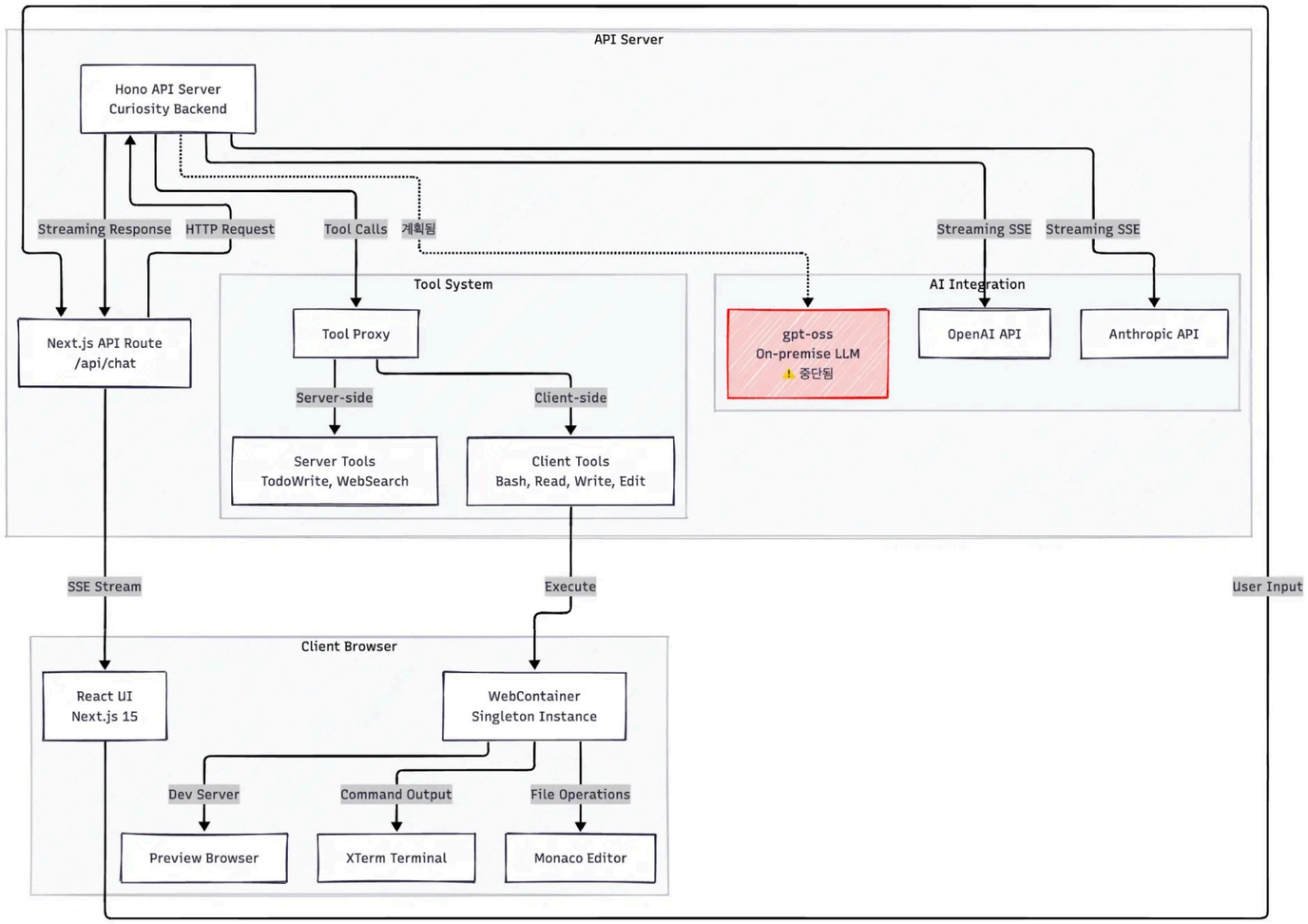
기존 AI 코딩 도구의 한계

- 로컬 환경 설정 필수: Node.js, Git, IDE 등 진입장벽
- 클라우드 API 의존: 외부 API 비용 및 데이터 보안 우려
- 수동 도구 실행: AI 제안 명령어를 직접 실행해야 함
- 실시간 피드백 부족: AI 응답 완료까지 대기

"비개발자도 브라우저에서 즉시 사용 가능하고, 온프레미스 환경에서 데이터 보안을 지키며, AI가 자동으로 MVP를 구축해주는 플랫폼이 필요함"

🏗️ 큐리오시티 설계

요구사항과 해결 방안



1. 비개발자도 즉시 사용 가능한 환경

→ WebContainer API 기반 브라우저 내 Node.js 환경 구축

- 로컬 설치 없이 브라우저만으로 즉시 개발 환경 제공
- 전역 싱글톤 패턴으로 WebContainer 인스턴스 관리

2. 데이터 보안을 위한 온프레미스 배포

→ 자체 호스팅 가능한 API 서버 설계

- Hono 기반 경량 API 서버로 온프레미스 배포 용이
- 환경변수 기반 AI 프로바이더 설정

3. 실시간 AI 응답으로 빠른 피드백

→ SSE 기반 스트리밍 응답 시스템 설계

- AsyncGenerator 패턴으로 토큰 단위 실시간 스트리밍
- 부분 응답과 완료 응답 분리로 즉각적인 피드백

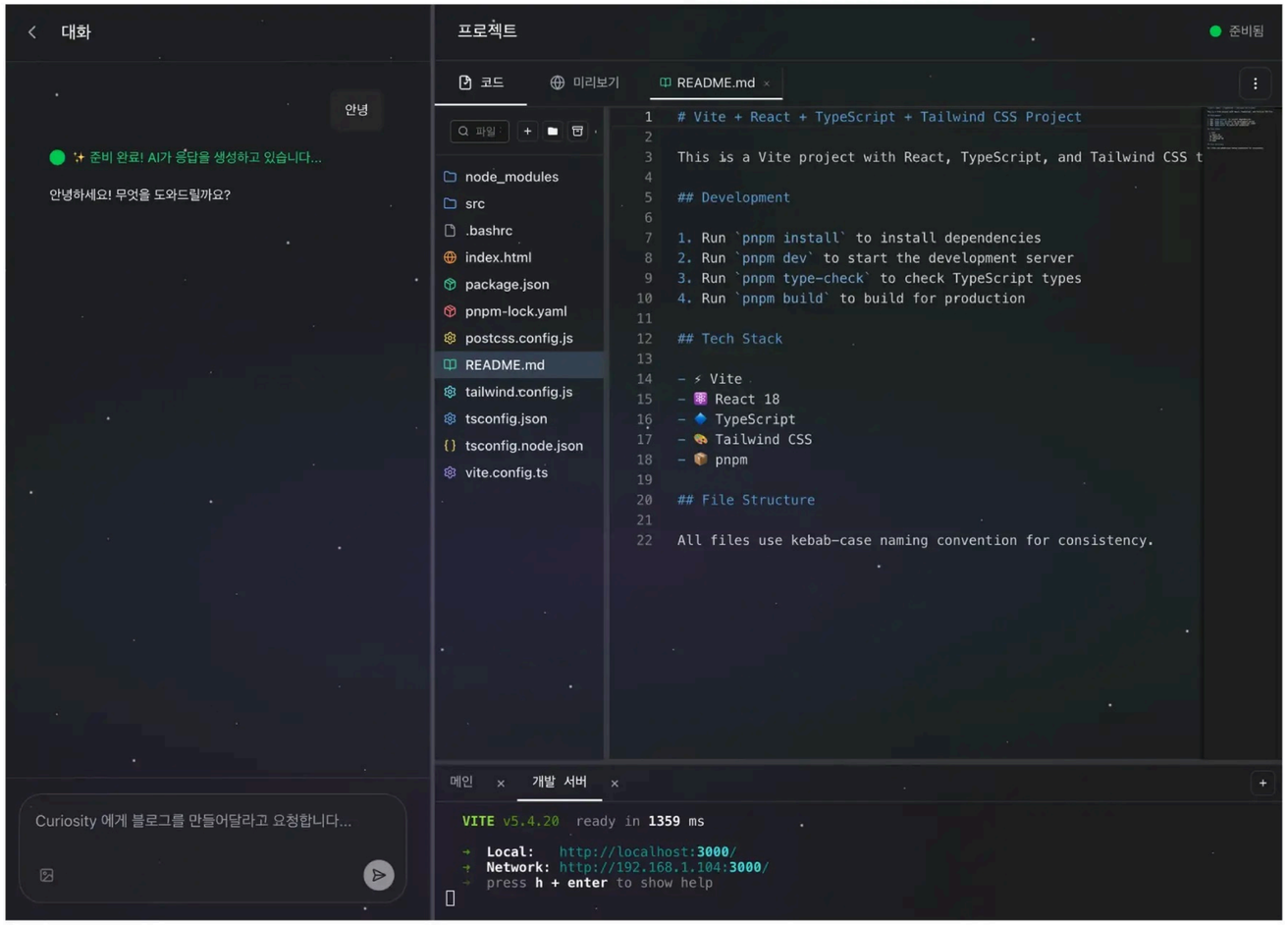
4. AI가 제안한 작업 자동 실행

→ 재귀적 도구 실행 엔진 설계

- API 호출 → 도구 실행 → API 호출 무한 반복 구조
- 반복 패턴 감지 알고리즘으로 무한 루프 방지

큐리오시티 구현

기술적 도전과 해결



1. WebContainer 통합 및 파일시스템 프록시

- 전역 인스턴스 관리: 여러 컴포넌트에서 동일한 WebContainer 재사용
- 다중 경로 시도 로직: /path, path, ./path 등 4가지 형식 자동 시도
- 커스텀 명령어 지원: .curiosity 폴더에 스크립트 배치

2. 실시간 스트리밍 응답 처리

- AsyncGenerator 기반 스트리밍: OpenAI/Anthropic API 청크 단위 응답
- 부분 메시지 누적: accumulatedText로 토큰 누적 및 실시간 UI 업데이트
- 비용 및 시간 추적: 최종 응답에서 토큰 사용량 기반 비용 계산

3. 재귀적 도구 실행 자동화

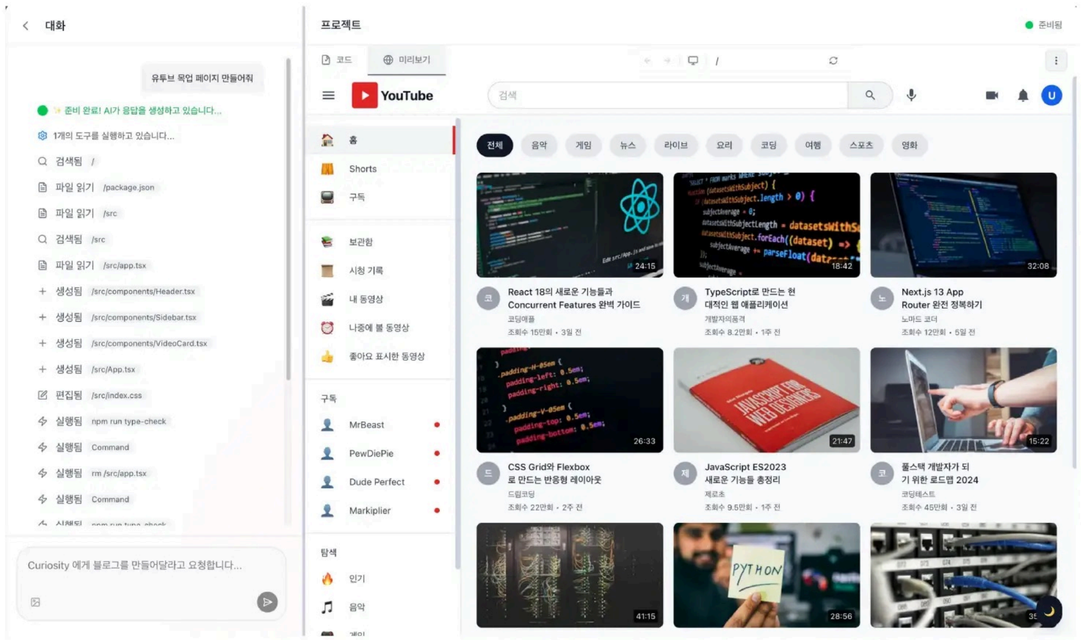
- 무제한 깊이 재귀: processMessageRecursively(depth) 함수로 도구 체인 자동 실행
- 반복 패턴 감지: 최근 5회 히스토리 추적, 동일 패턴 3회 반복 시 중단
- 개별 도구 타임아웃: 각 도구 실행에 30초 타임아웃 적용

기술적 의사결정

- 빠른 MVP vs 완벽한 설계: 재귀 패턴으로 복잡도 감소
- 성능 vs 안정성: 타임아웃과 반복 감지로 안정성 우선
- 클라우드 vs 온프레미스: 데이터 보안을 위해 온프레미스 우선 설계

🚀 큐리오시티 결과

비즈니스 임팩트 및 성과



0초
환경 설정 시간
브라우저만으로 즉시 시작

실시간
AI 응답 표시
토큰 단위 스트리밍

95%+
파일 접근 성공률
다중 경로 시도

프로젝트를 통해 배운 점

아키텍처 설계

- WebContainer 싱글톤 패턴으로 리소스 효율성 극대화
- 재귀 구조로 복잡한 비즈니스 로직을 단순하게 표현
- 클라이언트-서버 도구 분리로 확장 가능한 구조 설계

안정성과 사용성의 균형

- 무한 재귀의 위험성을 패턴 감지로 해결
- 개별 도구 실패가 전체 플로우를 중단시키지 않도록 설계
- 비개발자도 이해할 수 있는 직관적인 피드백 제공

아쉬운 점 및 중단 사유

프로젝트 중단 (2025년 9월 12일)

- 온프레미스 LLM 통합 작업 미완료
- 외부 API 의존성 제거 목표 미달성
- 라미앱 프로젝트로 리소스 집중

향후 개선 방향 (중단 전 계획)

- 온프레미스 LLM 서버 통합
- 폐쇄망 환경 완전 독립 실행
- 도구 실행 병렬화