이지훈

Email. jihoon7705@gmail.com
GitHub. jiji-hoon96
Blog. hooninedev.com

단일 클리닉 환자 12명이 쓰던 환자 건강정보 관리 서비스를,
캘리포니아 **클리닉 46개, 환자 4,000+명**이 쓰는 **Medicare RPM/CCM 청구 플랫폼으로** 확장했습니다.
사용자가 겪는 시간으로 트레이드오프를 판단하고, 그 근거를 구조에 남겨 팀의 품질로 만듭니다

HicareNet R&D 개발 그룹 · 2023.03 ~ 재직중

만성질환 환자가 집에서 측정한 정보를 미국 클리닉 의료진과 담당 careteam이 원격으로 관리하고, 그 기록으로 Medicare 진료비를 청구하게 하는 서비스입니다.

프로덕트 개발

환자 등록과 동의서 처리, EMR 데이터 연동, 보험 자격 확인, 청구 리포트 생성까지 전 구간 담당

React 19 · TypeScript 7 · Vite 8(rolldown) · React Query · Zustand 5 · Jotai · MUI 7 · Emotion · react-hook-form · zod · i18next · @react-pdf/renderer · Vitest · Playwright

RPM/CCM 2023.07 ~

프런트엔드 3인 중 리드 · 설계와 구현 주도

· **원격 환자 모니터링 청구 시스템의 메인 개발을 담당**해, 실시간 이상징후 알림과 관리 시간 기록의 소켓을 Worker 하나 안의 연결 계층으로 모으고 성격에 따라 lock 과 구독으로 갈라 소켓 수명을 화면 렌더 주기와 분리. 관리 시간이 곧 청구 요건이라 점유를 한 사람으로 제한하되 시도한 사람에게는 환자 정보와 모니터링 현황을 읽기 전용 모달로 열어, **관리 시간 이중 기록을 막으면서 조회 업무는 멈추지 않게 설계**

· 표로만 제공되는 측정 데이터를 의료진이 읽기 어렵겠다는 의문에서 Sentry session replay 로 클릭과 스크롤 경로를 보고, tag 와 context 로 그 구간의 체류 시간만 갈라 재서 평균 21초를 확인. 표 하나였던 화면을 측정 항목별 멀티 차트와 커스텀 표로 바꿔 기간과 표시 항목을 골라 비교하게 개선, **평균 체류 시간 21초에서 7초로 67% 단축**

· 월 3,000건 이상 나가는 Medicare 청구 리포트가 통과하는 화면에서, 부분 실패한 EMR 일괄 업로드를 같은 파일로 다시 올려 중복 청구가 되던 구조를, 형식은 스키마로 서버 요청 전에 걸러 통과한 건만 보내고 데이터 정합성은 서버가 판정해 결과를 건별로 돌려주도록 분리. 실패한 건만 다시 모아 재시도하게 해 **서버 검증 비용을 줄이고 사람이 되돌릴 수 없는 중복 청구 경로를 차단**

· RPM 단독이던 제품에 만성질환 관리(CCM) 청구를 얹기 위해 ICD-10 진단 코드를 축으로 케어플랜 설정과 월별 목표 관리, CCM 추적 관찰(F/U) 플로우를 구현, **CCM 을 취급하는 클리닉 12곳 신규 계약**

· 14개월간 레거시로 남아 있던 대시보드를 클리닉이 간결하게 보고 싶다고 요구했으나 무엇을 올릴지는 정해지지 않아, Sentry 에서 반복 조회가 많은 항목을 찾아 **기획자와 함께 대시보드로 다시 설계·구현**. RPM/CCM 을 갈라 측정 현황과 청구 조건 충족 비율을 올려 그 달 청구를 못 채울 환자를 달 중에 미리 알 수 있게 개선, **일평균 조회 180회에서 900회 이상으로 5배 증가**

· 클리닉이 환자 명단을 수십 명씩 엑셀로 넘기는데 의료진이 화면에 하나씩 옮겨 적던 것을 일괄 업로드로 전환. 잘못 들어가면 환자 수십 명을 하나씩 지워야 하므로 zod 스키마로 행별 검증한 결과를 미리보기 표로 먼저 보여 사용자 확인 단계를 만들고, **같은 스키마를 개별 등록 폼과 공유해 두 경로의 검증 규칙이 갈리지 않게 통일**

· 환자 활성화가 등록과 서비스 선택, 동의서와 부속 파일 업로드로 나뉘어 각각 다른 화면에 있었는데, Sentry 로 사용자 경로를 보고 업무에서는 이것들이 하나로 이어진다는 것을 확인해 단일 흐름으로 병합, **활성화 시간 4분 16초에서 2분 11초로 49% 단축**

· 중환자로 바뀌거나 병원을 옮겨 suspended 로 넘어갈 때 상태값은 코드로 남지만 의료진이 그 판단을 기록할 자리가 없던 것을 프로세스 문제로 진단해, 작성자와 시각이 함께 남고 기간과 키워드로 조회되는 환자 이력을 기획에 건의하고 구축. 이 기록을 dashboard 와 claim 이력 관리가 다시 쓰게 해 **여러 의료진이 같은 환자의 경과를 각자 다르게 알던 것을 하나의 기록으로 통일**

Tech Admin 2023.09 ~

프런트엔드 단독 · 설계와 구현 전 구간 담당

· 담당 클리닉과 직군, 역할에 따라 RPM/CCM 서비스의 접근 권한이 유동적으로 바뀌는데 그 변경을 개발자가 직접 고치고 있던 것을, 담당 매니저가 어드민에서 관리하고 신규 인원의 권한을 등록하는 폼까지 구현, **인원과 역할이 바뀔 때 담당자에서 대응할 수 있게 개선**

· 장비가 박스로만 오기도 하고 청구서와 함께 엑셀로 오기도 하는데 둘 다 담당자가 하나씩 옮겨 적어 시간과 인건비가 들던 것을 발견해, 박스는 IMEI 를 사진으로 찍어 OCR 로 텍스트를 뽑아 등록하고 엑셀은 스키마 검증 뒤 일괄 등록하도록 입력 형태마다 갈라 자동화. 기기 코드 서버 조회를 등록의 전제로 넣어 잘못 읽힌 값이 재고에 들어가는 경로를 닫고, **장비 담당자 월 공수 10시간에서 1시간 20분으로 87% 절감**

· 클리닉마다 다른 CPT 단가와 수익 배분율을 적용하여 월별 인보이스를 생성하고, 사용자가 단가나 청구 금액을 고치면 표의 나머지 항목이 함께 조정되게 구현. 금액이 직접 걸린 경로라 표의 값을 클라이언트와 서버에서 갈라 검증하고 도메인·비즈니스 로직을 E2E 테스트로 검증해, **단가 관리부터 보험사 제출용 파일 생성까지 한 흐름으로 잇고 금액 산출을 회귀 테스트로 고정**

HRA(건강위험평가) 설문 플랫폼 2025.09 ~

미국 Medicare Advantage 가입자 대상 건강위험평가(HRA) 플랫폼 · 프런트엔드 설계·구현과 기획·디자인, 관계사 커뮤니케이션 담당

· 앞 답변에 따라 필수와 활성 여부가 바뀌는 건강위험평가 설문을 다중 폼 50+개로 구성하고, 조건부 규칙을 컴포넌트 분기로 흩지 않고 스키마 한 곳에 모아 **규칙이 개정될 때 고칠 자리를 한 곳으로 고정**

· 폼 제출과 서명을 DocuSign 으로 병합해 파일을 만들던 과정이 번거로웠고, 문항을 정하는 주체가 보험사라 계약이 늘 때마다 코드로 문항을 확장하는 데 한계가 있었음. 보험사가 만든 템플릿을 그대로 PowerForm 에 입히는 방식으로 전환해 **신규 보험사 계약과 다국적 가입자를 코드 변경 없이 받을 수 있게 개선**

· 상담원이 여러 언어권 가입자를 번갈아 응대하는 업무 조건에서 i18next 로 언어당 번역 키 226개를 6개 언어 전부 작성, **HRA 진행 환자 600+ 확보**

성능과 구조 개선

기능을 확장하며 만난 병목을 지표로 특정하고, 라이브러리 트레이드오프를 PoC 로 검증해 개선

라이브러리 트레이드오프와 병목 개선

· 두 서비스에 jQuery 와 moment, lodash 가 남아 있었고 정적 import 로 걸려 있어 로그인 화면 하나를 열어도 전부 내려왔음. moment 와 lodash 를 사내 공용 라이브러리로 이관하고 router 를 인증 상태로 분기해 미인증 경로에는 로그인 화면만 존재하게 재구성, **초기 로딩 전송량 gzip 2,303kB 에서 829kB 로 64%, datetime vendor 청크 220.78kB 에서 27.31kB 로 88% 감소**

· LESS 와 CSS 파일이 두 개로 분기되어 빌드에 연결되지 않아 수정이 반영되지 않고, 앱 루트 스타일시트가 정적으로 적재되던 것을 Emotion 으로 모으고 외부 CDN 의 TTF 에서 자체 도메인 woff2 subset 으로 교체, **렌더를 막는 스타일시트 gzip 209kB 에서 32kB, 첫 화면 폰트 전송 485kB 에서 171kB 로 감소**

· DOM 에 직접 붙이고 떼는 명령적 방식이라 종료 결과를 콜백 체이닝으로 받았고, modal 을 정적 import 하는 registry 가 항상 마운트돼 열지 않은 modal 의 vendor 까지 초기 그래프에 들어갔음. 생명주기를 선언적으로 다루는 라이브러리로 이관하고 registry 를 동적 로딩으로 전환해 entry 를 736kB 에서 468kB 로 줄였으나, 번들 지표만으로는 오픈 지연이 드러나지 않아 청크 도착과 DOM 반영 시각을 따로 재서 원인이 Suspense 의 300ms 교체 지연임을 확인. 여는 시점에 미리 로드해 직접 렌더하도록 바꿔 **번들 이득을 유지한 채 확인 modal 326.9ms 에서 21.0ms, 폼 modal 365.8ms 에서 61.1ms 로 단축**

· ESLint 와 Prettier 가 함께 돌면서 포매팅 규칙이 서로 다투고, ESLint 는 설치되지도 않은 config 를 extends 해 규칙이 조용히 무력화된 상태였음. 린트와 포맷을 Biome 하나로 합쳐 판정 주체를 단일화하고, **관련 개발 의존성을 두 서비스에서 각각 12종에서 1종, 10종에서 1종으로 축소**

개발자 경험 향상

반복에 드는 비용을 지표로 관측하고, 동료들의 불편을 넘기지 않고 그 비용을 제거

Biome · Sentry · Bitbucket Pipelines · Docker · AWS CodeDeploy · Playwright

사내 공용 패키지

· 사내 서비스가 DevExtreme 기반의 정형화되지 않은 UI 를 각자 갖고 따로 관리하던 것을 MUI 기반 디자인 시스템으로 리팩터링하고, 아이콘과 폰트, 토큰까지 라이브러리에서 함께 관리. 컴포넌트마다 Storybook 스토리를 붙여 기획·디자인이 같은 화면을 보고 논의하게 해 **신규 프로젝트의 개발 속도와 기존 프로젝트의 디자인 구조 정합성을 높임**

· 프런트엔드와 백엔드가 같은 도메인 규칙을 각자 구현하고 있어 한쪽만 고치면 조용히 갈라지던 것을, **공용 유틸리티 라이브러리를 새로 만들고** Docusaurus 로 문서 사이트를 세워 버전과 문서를 함께 관리

품질과 관측

· 두 포털이 각각 개발과 운영을 갖는데 회귀를 잡는 장치가 없어 배포 뒤 무엇이 깨졌는지를 사용자가 알려주던 것을, 도메인 로직 유닛 테스트를 먼저 세우고 그 시나리오를 Playwright·Docker 4개 환경 매트릭스의 매일 자동 실행으로 전환, **QC 회귀 확인을 하루 40분 이상, 정기 배포일 1시간 이상 단축**

· 사용자가 미국에 있는데 한국에서 문제 인지부터 QA·기획 전달과 QA 재현, 이슈 할당, 개발자 재현을 거쳐야 해결에 닿던 것을, Sentry 에 수집만 되고 보는 자리가 없던 데이터 위에 대시보드 위젯 10개와 지표 임계 알림 12개, 태그 체계 12종을 세워 전환. 사용자가 문제를 알리면 개발자가 시각과 태그로 session replay 를 찾아 그 자리에서 재현하고 테스트 코드로 조건을 남긴 뒤 고치게 해, **문제 인지부터 해결까지 총 평균 5.3시간에서 1.2시간으로 77% 단축**

빌드와 배포

· 자산 파일명이 고정이고 배포 시각을 쿼리스트링으로 붙이는 방식이라 쿼리가 바뀔 때마다 캐시가 무효화돼, 변경되지 않은 vendor 청크까지 매 배포마다 전량 다시 받고 있었음. 콘텐츠 해시로 전환해 바뀐 파일만 받게 하고, 파일명이 바뀌면 이미 열린 탭이 옛 청크를 404 로 받으므로 vite:preloadError 를 관측 도구에 보고한 뒤 사용자 확인을 거쳐 다시 불러오는 경로까지 구축, **같은 vendor 청크의 재다운로드 197kB·492ms 를 304 응답 102ms 로 되돌려 79% 단축**

· PuTTY 와 WinSCP 로 빌드 폴더를 서버에 직접 올려 배포하던 것을, build·upload·배포·health check·알림을 단계로 세우고 실패가 알림으로 돌아오게 만들어 사람이 순서를 놓치는 경로를 제거. 클라우드 인증은 장기 access key 를 심지 않고 OIDC 로 파이프라인의 신분을 증명해 IAM role 을 위임받는 방식으로 구성, **배포 시간이 dev 14~16분에서 6~8분으로 약 53%, prod 21~23분에서 8~10분으로 약 59% 단축**

AI 활용 경험

트렌드를 모두 follow up 하기보다 팀에 필요한 것을 가려 매주 공유·사내 발표하고, 생산성과 안정성 기준으로 적용

LLM 문서 자동화

· 공용 라이브러리는 소비하는 쪽이 문서만 보고 쓸 수 있어야 공용화 효과가 나는데, 문서 갱신이 손으로 하는 일이라 함수가 늘고 시그니처가 바뀌는 속도를 따라가지 못했음. 손으로 쓴 참조 문서를 출력 형식 기준으로 먼저 정한 뒤 git diff 로 바뀐 소스만 골라 Amazon Bedrock 으로 생성해 문서 사이트가 그대로 읽는 형태로 정제하고, 같은 방식으로 버전별 변경 목록을 만들어 릴리스 시점에 사내 메신저로 통지, **형식을 사람이 먼저 정하고 자동화가 따르게 해 사내 지식의 정본을 문서로 확보**

hicare-ai

· 사내 라이브러리와 컨벤션, 도메인 지식이 문서로는 있었지만 찾아 읽는 시간이 들고 동료마다 적용하는 편차가 컸음. 각각을 AI 스킬로 정의해 사내 레포 하나로 모으고, 함께 쓸 오픈소스 스킬의 정보를 취합해 활용할 수 있게 구성, **동료가 정보를 찾는 시간과 AI 활용 지식의 편차를 줄임**

규약 문서와 AI 리뷰 봇

· AI 로 코드를 쓰는 속도가 올라가면서 리뷰가 사람 시간에 묶이고 같은 지적이 반복되는데 그 지적의 근거가 사람마다 달랐음. 규약 문서를 먼저 쓰고, Amazon Bedrock 리뷰 봇이 그 문서와 앞서 정의한 스킬에서 라이브러리·도메인 정보를 함께 읽어 판정하게 구성. 변경 라인 주변과 연관 파일을 ripgrep 으로 발췌해 토큰 예산 안에 넣어 연관 관계까지 보고 1차 리뷰, **사람마다 갈리던 지적 근거를 문서와 스킬로 고정하고 반복되는 지적을 봇이 먼저 잡게 전환**