Jihoon Lee

Frontend Engineer

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

Grew a patient health-record tool used by **12 patients at one clinic** into a Medicare RPM/CCM billing platform serving **46 California clinics and 4,000+ patients**.

I decide trade-offs by the time users actually lose, and leave that reasoning in the architecture so it becomes the team's quality bar rather than my own.

HicareNet R&D Group

Mar 2023 – Present

A service where US clinic providers and care teams remotely manage readings chronic-disease patients take at home, and bill Medicare from that record.

Product Development

Owned the full span — patient onboarding and consent, EMR data integration, insurance eligibility checks, and claim report generation.

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 Jul 2023 – Present

Lead of 3 frontend engineers — drove architecture and implementation

- Led the core build of the remote patient monitoring billing system. Collapsed the real-time vital-alert and care-time-tracking sockets into a single connection layer inside one Worker, split by semantics into lock and subscription, so socket lifetime is decoupled from the render cycle. Because logged care time *is* the billing requirement, occupancy is limited to one user — but anyone else who tries gets a read-only modal of patient data and monitoring status, **preventing double-counted care time without stalling lookup work**.
- Suspected clinicians could not read measurement data served as a bare table. Traced click and scroll paths in Sentry session replay, isolated dwell time for that segment by tag and context, and measured a 21s average. Replaced the single table with per-metric multi-charts and a configurable table for choosing periods and columns — **average dwell 21s → 7s (67% faster)**.
- On the screen pushing **3,000+ Medicare claim reports a month**, partially failed EMR bulk uploads were re-submitted as the same file and became duplicate claims. Separated the two concerns: schema validation filters format before the request, and the server adjudicates data integrity and returns per-record results, so only failed records are retried. Cut server validation cost and closed an **irreversible duplicate-billing path**.
- Extended an RPM-only product into Chronic Care Management (CCM) billing — care-plan setup, monthly goal tracking and CCM follow-up flows built around ICD-10 diagnosis codes as the axis — **12 new clinic contracts**.
- A dashboard had sat as legacy for 14 months; clinics wanted it simpler but could not say what belonged on it. Mined Sentry for the most repeatedly queried items and redesigned it with the PM. Splitting RPM and CCM to surface measurement status and billing-criteria completion lets clinics see mid-month which patients will miss billing — **daily views 180 → 900+ (5×)**.
- Clinics hand over patient rosters in Excel dozens at a time and clinicians retyped them one by one; converted this to bulk upload. Since a bad import means deleting dozens of patients by hand, row-level zod validation results are shown in a preview table as an explicit confirmation step, and the same schema is shared with the single-registration form so **the two paths cannot drift apart**.
- Patient activation was scattered across separate screens for registration, service selection, consent and attachment upload. Sentry user paths showed these were one continuous job in practice, so I merged them into a single flow — **activation time 4m16s → 2m11s (49% faster)**.
- When a patient turned critical or changed hospitals, the `suspended` state was recorded in code but clinicians had nowhere to record the judgment behind it. Diagnosed it as a process gap rather than a UI one, proposed it to planning, and built a patient history log carrying author and timestamp and queryable by period and keyword. Dashboard and claim history now read the same log, so **clinicians no longer hold different versions of the same patient's course**.

Tech Admin Sep 2023 – Present

Sole frontend engineer — full design and implementation

- RPM/CCM access rights shift with assigned clinic, job function and role, and developers were editing them by hand on request. Moved ownership to an admin console the responsible manager operates, including the form for registering permissions for new staff, so **staffing and role changes are handled by the people who own them**.
- Devices arrive either as boxes or as Excel attached to invoices, and both were retyped entry by entry at real time and labor cost. Automated per input shape: box IMEIs are captured by photo and read through OCR, Excel is schema-validated and bulk-registered. Made a server lookup of the device code a precondition for registration, closing the path where a misread value enters inventory — **monthly workload 10 hours → 1 hour 20 min (87% reduction)**.
- Built monthly invoicing across per-clinic CPT rates and revenue-share ratios, where editing a rate or billed amount re-adjusts the remaining rows. Because money rides directly on this path, table values are validated separately on client and server and domain/business logic is covered by E2E tests — connecting rate management through to payer-submission file generation in one flow and **pinning amount calculation with regression tests**.

HRA Survey Platform Sep 2025 – Present

Frontend architecture and implementation, plus planning, design and partner communication

- Built a Health Risk Assessment survey for US Medicare Advantage members as **50+ interlinked forms** whose required and enabled states depend on prior answers. Conditional rules live in one schema instead of being scattered across component branches, **fixing a single place to edit when the regulation is revised**.
- Merging form submission with DocuSign signing to produce files was cumbersome, and since payers own the questions, extending them in code did not scale with each new contract. Switched to applying payer-authored templates directly through PowerForm, so **new payer contracts and multilingual members are onboarded with no code change**.

- Agents serve members across language groups back to back; authored all **226 translation keys per language across 6 languages** with i18next — **600+ patients completed through HRA**.

Performance & Architecture

Pinned bottlenecks found while scaling features to measurable metrics, and validated library trade-offs with PoCs before committing.

- jQuery, moment and lodash lingered across two services under static imports, so opening the login screen alone pulled all of it down. Migrated moment and lodash into the internal shared library and branched the router on auth state so unauthenticated routes contain nothing but the login screen — **initial payload 2,303kB → 829kB gzip (64% less)**, **datetime vendor chunk 220.78kB → 27.31kB (88% less)**.
- LESS and CSS had forked into two files, one never wired into the build, so edits silently did nothing, and the app root stylesheet was loaded statically. Consolidated onto Emotion and replaced external-CDN TTFs with self-hosted woff2 subsets — **render-blocking stylesheet 209kB → 32kB gzip**, **first-screen font payload 485kB → 171kB**.
- Modals were attached and detached imperatively on the DOM with close results returned through callback chains, and a registry statically importing every modal stayed permanently mounted, pulling vendors for never-opened modals into the initial graph. Migrated to a library handling lifecycle declaratively and made the registry dynamically loaded — **entry 736kB → 468kB**. Bundle metrics alone did not expose the open latency, so I measured chunk arrival and DOM commit as separate timings and identified Suspense's 300ms swap delay as the cause; preloading at open time and rendering directly kept the bundle gains — **confirm modal 326.9ms → 21.0ms, form modal 365.8ms → 61.1ms**.
- ESLint and Prettier ran together with formatting rules fighting each other, and ESLint extended a config that was never installed, leaving rules silently disabled. Unified lint and format onto Biome as the single arbiter — **dev dependencies cut 12 → 1 and 10 → 1 across the two services**.

Developer Experience

Measured what repetition actually costs, and removed that cost rather than letting a colleague's friction pass.

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

Internal shared packages

- Internal services each carried their own unstructured DevExtreme UI and maintained it separately; refactored into an MUI-based design system managing icons, fonts and tokens inside the library. A Storybook story per component let planning and design argue over the same screen — **raising delivery speed on new projects and design consistency on existing ones**.
- Frontend and backend implemented the same domain rules independently and diverged silently whenever only one side changed. Built a shared utility library and stood up a Docusaurus site so **versions and docs are managed together**.

Quality & observability

- Two portals each run dev and prod with no regression safety net — users were the ones telling us what broke after a deploy. Established domain-logic unit tests first, then converted those scenarios into daily automated runs on a **Playwright + Docker 4-environment matrix** — **40+ min/day and 1+ hour per release day** saved on QC regression checks.
- Users are in the US while the team is in Korea, so a fix travelled through detection, QA/planning handoff, QA reproduction, issue assignment and developer reproduction. Sentry was collecting data nobody had a place to read, so I built **10 dashboard widgets, 12 threshold alerts and a 12-type tag scheme** on top of it. A developer now finds the session replay by timestamp and tag, reproduces it on the spot and pins the condition in a test before fixing — **detection to resolution 5.3h → 1.2h (77% faster)**.

Build & deployment

- Asset filenames were fixed with the deploy timestamp appended as a query string, so every deploy invalidated the cache and re-downloaded even unchanged vendor chunks. Moved to content hashing so only changed files transfer; since changing filenames makes already-open tabs 404 on old chunks, I also built the path that reports `vite:preloadError` to observability and reloads after user confirmation — **re-download of the same vendor chunk 197kB / 492ms → a 304 in 102ms (79% faster)**.
- Deploys were folder uploads pushed to the server by hand over PuTTY and WinSCP. Staged build, upload, deploy and health-check notification, with failures routed back as alerts, removing the path where a person skips a step. Pipelines prove their identity via OIDC and assume an IAM role instead of embedding long-lived access keys — **deploy time dev 14–16m → 6–8m (~53%), prod 21–23m → 8–10m (~59%)**.

Working with AI

I filter for what the team actually needs rather than following every trend, share it weekly and present internally, and adopt on productivity and stability grounds.

- **LLM documentation automation** — A shared library only pays off if consumers can use it from the docs alone, but hand-written updates could not keep pace with functions being added and signatures changing. Fixed the output format of the reference docs first, then generated only the sources changed per `git diff` through Amazon Bedrock into a shape the docs site reads directly; the same pipeline builds per-version changelogs announced to the internal messenger at release. **Humans set the format and automation follows** — which is what made the docs the canonical source of internal knowledge.
- **hicare-ai** — Internal libraries, conventions and domain knowledge existed as documents, but finding and reading them cost time and every colleague applied them differently. Defined each as an AI skill, gathered them into one internal repo, and aggregated information on open-source skills to use alongside — **cutting both search time and the variance in how colleagues apply AI**.
- **Convention docs & AI review bot** — As AI raised the rate code was written, review became bound to human hours and the same remarks repeated with different justifications per reviewer. Wrote the convention document first, then had an Amazon Bedrock review bot judge against it plus the skills above, excerpting changed lines and related files with ripgrep to fit the token budget so first-pass review sees the relationships too — **grounds for review comments fixed in docs, repetitive ones caught by the bot**.

Pai Chai University Bachelor of Arts, English Language and Literature **Mar 2014 – Feb 2020**