

Vorda — building a full NDIS compliance platform, solo, in under two months

Senior Frontend Engineer · Web admin + worker mobile app + landing site · 2026

< 2 months END-TO-END BUILD	1 engineer SOLE FRONTEND OWNER	3 surfaces ADMIN · MOBILE · LANDING	2 platforms WEB + NATIVE, ONE TYPE SYSTEM
--	--	---	---

CONTEXT

Vorda is an NDIS (Australian disability-services) provider that needed a complete digital platform to run its operations — a web admin app for office staff to manage participants, workers, rostering, payroll and compliance, plus a mobile app for support workers in the field. I joined as the sole frontend engineer and built the admin, mobile, and marketing surfaces from an empty repository.

THE CHALLENGE

- Deliver an entire multi-surface platform — web admin, worker mobile app, and marketing site — on a ~2-month timeline, as the only frontend engineer.
- The backend was being built in parallel, so most endpoints weren't ready — or changed shape — while I was building the UI that depended on them.
- The domain is compliance-heavy: rostering, participant plans, worker credentials, incident reporting and payroll all carry strict rules, so correctness mattered as much as speed.

APPROACH & KEY DECISIONS

Mock-first architecture

Rather than block on the backend, I built against Mock Service Worker mocks shaped to the expected contract, gated behind per-section feature flags. Each module could be developed and demoed fully mocked, then flipped to live staging data one flag at a time — with no rewrites. This decoupled my delivery from backend readiness and kept progress continuous.

A shared, strongly-typed API contract

All request/response types lived in a shared TypeScript package consumed by both web and mobile. When the live API drifted from the documented contract — which it often did — the mismatch surfaced at compile time instead of as a production runtime bug, and the two apps stayed in lockstep.

Verifying against reality, not docs

Because documented endpoints frequently differed from what staging actually returned, I probed live endpoints directly to confirm real shapes before wiring them in, and reported divergences back to the backend team with specifics — catching missing fields and wrong types before they reached users.

Secure AI integration

The platform uses an AI service to generate progress notes and analyse incident reports. I integrated it through a server-side proxy so the API key never shipped in the client bundle — keeping the secret off the device while still

delivering the feature in-app.

WHAT I BUILT

- **Web admin** — participant onboarding, worker credential & document management, rostering/scheduling, payroll with Xero synchronisation, invoicing, and incident/compliance reporting.
- **Worker mobile app** — the field-facing app for support workers: shifts, participant details, document upload, and incident/progress-note capture with biometric-secured sign-in.
- **Landing site** — the public-facing marketing site.
- **Shared foundation** — typed API client, auth store, normalised error handling, and the shared types package underpinning both apps.

Stack: Next.js 16 · React 19 · React Native (Expo SDK 55) · TypeScript · Tailwind v4 / NativeWind · TanStack Query v5 · Zustand · react-hook-form + Zod · MSW · EAS

OUTCOME

Delivered a complete, multi-surface platform — web admin, worker mobile app, and landing site — solo, in under two months. The mock-first + typed-contract approach meant features were built, demoed and reviewed continuously despite a backend under active construction, and each section went live by flipping a single flag once its endpoints were ready. Internal test builds shipped to stakeholders throughout via EAS.

WHAT THIS PROJECT SHOWS

Senior ownership of an ambitious build under real constraints: architecting for a moving backend, enforcing type-safety across web and native, integrating third-party and AI services securely, and shipping a broad, compliance-heavy product fast — without trading away correctness.