

Medix

Designing and Building a Multi-Tenant Clinic Management Platform

Baran Akkanat

Creator and sole engineer — product, architecture, implementation and operations

Development period: 16 June – 12 September 2026 · Document revised September 2026

About this document, and how to read its claims

Every factual statement in this case study is grounded in the Medix repository: source code, the Prisma schema, SQL migration files, architecture decision records, the test suite, CI workflow definitions, deployment configuration and Git history. Functionality is labelled **IMPLEMENTED**, **PARTIAL** or **NOT PRESENT**, and nothing labelled partial or absent is described elsewhere as shipped.

Quantities were measured directly from the repository; the exact commands are listed in Appendix A so that any number here can be reproduced. Where a figure comes from the project's own status records rather than from a measurement I performed, it is attributed as such. Where an exact figure could not be established reliably, qualitative language is used instead.

Contents

1	Executive Summary	3
2	Problem Context	6
3	Product Vision and Design Principles	6
4	Core Product Workflows	8
5	System Architecture	11
6	Data Model and Multi-Tenancy	13
7	Key Engineering Decisions	15
8	Reliability, Privacy and Safety Engineering	20
9	Testing and Verification Strategy	22
10	Continuous Integration and Release Discipline	24
11	Selected Technical Challenges	26
12	Product and User-Experience Decisions	29
13	Current Status: Implemented, Partial, Absent	30
14	Roadmap	30
15	What I Learned	33
16	Technology Stack	34
17	Evidence and Verification	34
A	Measurement Methodology	36

1 Executive Summary

1.1 What Medix is

Medix is a **multi-tenant clinic operating system**: one deployment that serves many independent clinics, each with its own staff accounts, patients, appointment book, physical locations, service catalogue and patient conversations. It is written in TypeScript on Next.js (App Router) with PostgreSQL through Prisma, and is deployed as a container on Google Cloud Run.

It is not a generic role-and-permission SaaS application with a medical theme. Patient visibility is scoped by assignment and clinic policy rather than granted wholesale; every action is attributable to an individual staff member; and the separations that matter — between clinics, between clinical and operational roles, between platform operators and patient data — are enforced by database constraints rather than by convention.

1.2 The problem it addresses

The information a private clinic runs on is usually spread across systems that do not know about each other: the patient record in one place, the appointment book in another, the WhatsApp conversation on a staff member's personal phone, the reminders nobody has time to send, the consent form in a drawer, and no reliable answer to the question of who is allowed to see any of it.

Medix consolidates those into one system *without* collapsing the boundaries between them. A receptionist who can book an appointment still cannot open a clinical note. A visiting consultant sees only patients assigned to them, regardless of what the clinic's default visibility is set to. A platform operator can see that a clinic exists, is paying and has a healthy WhatsApp connection — and cannot read a single one of its charts.

1.3 Implemented scope

Patient records with duplicate detection, spreadsheet import, role-sanitised projections, export and a gated deletion lifecycle; scheduling with practitioner unavailability, multi-location support and database-enforced double-booking prevention; patient-facing self-service booking; WhatsApp messaging through a provider abstraction over Twilio and the Meta Cloud API; automated appointment reminders with delivery receipts and bounded retry; structured follow-up protocols and derived clinical recalls; deterministic guided clinic setup; role-based authorization across nine roles and 52 permissions; consent records, legal holds and a retention engine; an audited platform control plane with time-bound break-glass access; subscription entitlement convergence across Stripe, Apple and Google; and six user-interface locales including right-to-left Arabic. An Expo mobile client and an Electron desktop shell live in the same repository as clients of the same server.

1.4 Role and provenance

Creator and sole engineer — product, architecture, implementation and operations.

Every commit in the repository is authored by one person. The current line carries **642 commits**; **764** exist across all references. Development ran from **16 June to 12 September 2026**, with 390 commits in August and 224 in the first twelve days of September. I made the product decisions, designed the data and authorization models, wrote all 52 migrations, built the interface and the design system, wrote the test suite and the CI pipeline, and operate the deployment. Section 17 sets out how this can be verified independently despite the repository being private.

1.5 Deployment status, stated precisely

What “production” means here — and what it does not

Medix is deployed to a **production infrastructure environment** on Google Cloud Run and is served at medix.healthcare, which responded HTTP 200 when checked on 14 September 2026, as did its public `/account-deletion` and `/security` resources.

This does not mean that clinics or patients are using Medix. The environment currently carries a demonstration clinic and three empty test tenants. **There are no real patient records.** All data exercised to date — in development, in CI, in the demonstration workspace and in the deployed environment — is synthetic. The pilot topology described in the repository’s operations documentation, which isolates a real clinic’s database, storage bucket and encryption key from everything used for testing, exists as a written design; no pilot clinic has been onboarded.

Read every statement in this document accordingly: it describes a system built and operated to a production standard, not a system with clinical adoption.

1.6 Engineering depth beyond CRUD

Three properties distinguish this project from an application that merely stores and retrieves records. Each is developed in full later in the document; they are named here because they are the argument.

Invariants are expressed in PostgreSQL, not in convention. Cross-tenant references are made *unrepresentable* by 33 composite (`clinicId`, `id`) foreign keys rather than merely unlikely. Concurrent double-booking is refused by a GiST exclusion constraint rather than by an application check that loses the race. Idempotency across four different messaging paths is four unique indexes rather than four code paths somebody has to remember to write. (Section 7, decisions 1, 2 and 4.)

An architectural-invariant test layer. 111 test files read the source tree, the design-token stylesheet, the locale files and the CI workflow definitions and fail when a boundary is crossed — an encrypted storage key leaving the storage layer, an audit row being mutated anywhere under `src/`, a control-plane query touching a clinical column, a removed third-party model call reappearing. These are rules a type system cannot express and a code review will eventually miss. (Section 9.)

Documented refusals. Several of the strongest decisions in the codebase are things deliberately *not* built, with the reasoning recorded in an architecture decision record and the absence pinned by a test: automated recall outreach, a service worker, a language-model call in onboarding, media messaging, an “impersonate this clinic” capability in the control plane. Deciding what not to build, and then making the decision durable, is product engineering rather than restraint. (Sections 7 and 12.)

1.7 Project at a glance

Measure	Value	Note
Commits, current line	642	Single author; 764 across all refs
Development period	13 weeks	16 Jun – 12 Sep 2026
Application code, <code>src/</code>	88,546 lines	TypeScript/TSX; excludes the untracked generated Prisma client
Test code, <code>tests/</code>	71,996 lines	
Mobile client	9,901 lines	<code>apps/mobile</code> , TypeScript
Tracked files	1,339	
Prisma models / enums	56 / 51	3,018-line schema
SQL migrations	52	Each entry checksum-verified
Composite tenant foreign keys	33	From 24 models, onto 11 targets
Roles / permissions	9 / 52	Plus one legacy role alias
Server Actions / API routes	43 / 58	Plus 52 page routes
Domain modules, <code>src/lib</code>	225	
React components	162	Under <code>src/components</code>
Test files	324	316 Vitest, 8 Playwright
Declared test cases	≈3,599	In 958 <code>describe</code> blocks
CI / operations workflows	19	16 require a typed confirmation
Interface locales	6	≈2,900 keys each, incl. RTL Arabic
Architecture decision records	6	
TODO / FIXME in <code>src/</code>	0	Verified; decisions are recorded, not deferred in comments

Table 1: Repository measurements, re-verified on 14 September 2026. Appendix A gives the command behind each figure.

2 Problem Context

The operational problems Medix addresses are legible in what the schema chose to model, and — more tellingly — in what the code refuses to infer.

Patient information is fragmented across incompatible surfaces. A patient’s history is simultaneously an appointment book, a set of clinical notes, a photograph archive, a consent record and a WhatsApp conversation. Medix models all five against one patient row and *derives* a single chronological journey from them rather than storing a second, divergent timeline. The derivation is a pure function over data-transfer objects the page already loads, so it costs no additional query, column or permission.

Practitioners work in more than one place. A surgeon may operate at a hospital on Mondays, consult at a private clinic on Thursday afternoons, and not work Wednesdays. Modelling that as separate tenants would give each site its own patient database; modelling it as a single schedule would let the clinic book somebody at a site where they are not physically present. Medix models `Location` as a workplace owned by the clinic, and `UserLocation` as the join carrying that practitioner’s weekly pattern *at that workplace*. Availability is the equation *working hours* – *unavailability* – *external busy time* – *existing appointments*, and the absence of a `UserLocation` row is itself a refusal.

Patient communication is repetitive, and easy to get wrong. Reminding every patient the day before an appointment is real work that produces no clinical value when done by hand, and produces harm when done twice or sent to somebody who asked not to be contacted. Medix automates that single task, records what the provider proved about each message, and stops there.

“Everyone sees everything” is wrong; so is “only the treating doctor”. A real clinic sits between those. Medix implements three per-clinic visibility modes — `assigned_only`, `hybrid` and `clinic_wide` — with roles that remain assigned-only whatever the clinic setting, a per-patient sharing flag, and a supervising-doctor relationship for assistants. Scope is applied *inside* the query, not by filtering results afterwards.

Setup is done by people who are not engineers. A clinic administrator connecting WhatsApp should not need to understand webhook subscriptions or long-lived token exchange. The connection flow performs those steps itself, and reports honestly when one of them silently did not happen — because a connection that looks perfect and receives nothing is otherwise discovered when a patient writes and nobody sees it.

Administrative actions must be accountable to somebody. Deleting a patient, exporting a record, suspending a staff account, reading a held message as a platform operator: each is recorded, several refuse to proceed unless the record can be written first, and none of them is reversible by editing the record of it.

3 Product Vision and Design Principles

These principles are inferred from decisions actually taken in the code. None was declared in advance; each is stated here because it is visible in more than one place, which is what makes it a principle rather than a preference.

The server is the only authority. The mobile client makes no authorization decision and never imports server code; it renders data-transfer objects the server produced and depends on a mirrored wire contract that a parity test holds structurally identical to what the API actually returns. The web middleware describes itself, in its own comments, as a coarse first pass, with the binding check performed against the database rather than against a token claim.

FIGURE PLACEHOLDER — SCREENSHOT TO BE INSERTED

Today — the daily clinic view

Replace this frame with `\includegraphics.`

Figure 1: The home screen is specified as a command centre rather than an analytics dashboard: it answers “what is now, what needs attention, what is next” and is designed around actions rather than metrics. The action queue is framed as a queue to clear, with a completion count, because a number that only goes up is not something a clinic can act on. Privacy is one thin line at the bottom, deliberately — it is the only privacy surface on this screen.

Fail closed, without exception. An unset cron secret means *nobody*, not everybody — the earlier `if (secret && token !== secret)` shape is named in the source as the bug it was. A practitioner with no location assignment is offered at no location rather than at every location. A procedure is publicly bookable only when a clinic opts that specific procedure in. Photo storage refuses to choose a driver rather than silently falling back to a local disk that will not exist in the deployed environment.

Put the guarantee at the lowest layer that can express it. Where a rule can be a database constraint, it is one. Application checks remain, but their job becomes producing a better error message rather than being the guarantee.

Never guess on a patient’s behalf. An inbound message that could belong to two clinics is held for a human rather than routed. A patient’s preferred language stays null until they state one; it is not inferred from a phone country code, a name, or a staff member’s browser. Historical appointments are not assigned a location that was never recorded.

Automation must be narrow, visible and attributable. Automated messages and human messages live in different tables and are merged only for display, with the automated ones carrying no body — a template is not correspondence. A recall date is a clinical fact; sending it is a commercial act; the two look identical in a database and are not the same thing.

State what is true, including when it is unflattering. The in-product trust surface reports the audit trail as *configured* rather than *enforced*, because the database grants that would enforce it are written and not yet applied. A clinician who cannot distinguish “nothing is wrong” from “I am not permitted to look” will assume the first, and building for that distinction changed several screens.

One design system, enforced rather than documented. Every colour, spacing, radius, type and motion value is defined once. A guard test recomputes WCAG relative luminance from that file and fails on drift, and the specification document states explicitly that where it and the stylesheet disagree, the stylesheet wins and the document is the defect.

4 Core Product Workflows

Each workflow below is described in three parts: the user’s goal, the system’s behaviour, and the engineering consideration that makes the behaviour non-trivial.

4.1 Patient management

Goal. Create and find a patient without creating a second copy of one who already exists, and see everything known about them in one place.

Behaviour. Creating a patient runs a clinic-scoped duplicate check across phone number, email and name-plus-date-of-birth, and *reports* candidates with masked contact hints. It never blocks and never merges. The reasoning is recorded in the module itself: twins share a surname and a birthday, a family shares a phone line, and a clinic that cannot register the second sister because the first exists is broken in a worse way than a duplicate. Spreadsheet import is split into a pure planning function and an apply step that consumes the exact object the user approved; a row matching an existing patient is classified and skipped, because there is no update path in the module at all.

Engineering. Patient reads compose a Prisma `where` clause from role, clinic visibility mode, direct assignment, primary-doctor relationship, creator and supervising-doctor chain — the scope is part of the query, never a filter applied to results. A separate function strips clinical fields from a record before it reaches a role that may not read them. `Patient.notes` and `medicalHistory` are encrypted with AES-256-GCM at the application layer, with a versioned `enc:v1: envelope`.

The reason import has no update path is worth stating plainly, because it is a counter-intuitive product decision with an engineering justification: a clinic importing from a previous system is almost always importing a stale export. An “update” would therefore let the *old* record overwrite whatever the clinic has since corrected, silently, across hundreds of rows. Merging two records is a clinical judgement about one person’s history; a spreadsheet row does not get to make it.

FIGURE PLACEHOLDER — SCREENSHOT TO BE INSERTED

Patient record

Replace this frame with `\includegraphics`.

Figure 2: The record is a single derived journey rather than six tabs. Its design note records a real leak this shape prevents: a component typed as `Patient` was handed a spread of a row fetched with relations included, and React serialises the runtime object rather than the type annotation — so raw storage keys reached the client payload. The journey nodes carry photograph and consent *counts* rather than records, so there is no locator on this surface to leak.

4.2 Scheduling and multi-location

Goal. Book the right practitioner, at the right workplace, at a time they are genuinely free.

Behaviour. The schedule is calendar-first. Availability subtracts practitioner unavailability — a closed vocabulary of vacation, personal time, blocked hours, lunch and clinic closure — and busy time imported from a connected personal calendar. Location filters the grid but gates public booking absolutely.

Engineering. Three rules about location interact and each is written down rather than assumed. Appointments and external busy time are *not* location-scoped, because a person booked at one site is unavailable at every other site too. Unavailability is conditionally scoped: personal leave blocks

everywhere, a closed treatment room blocks only there. Bookability is a pair, so an unconfigured practitioner-location combination publishes nothing.

Times are wall-clock and resolved in the *clinic*'s timezone, never the browser's. Slot generation drops times that do not exist on a spring-forward day and are ambiguous on a fall-back day rather than resolving them to a guess. Schedule range queries are bounded (370 days, 10,000 rows) and keyset-paged with an overflow probe, so an oversized range raises a named error instead of silently truncating a clinician's day.

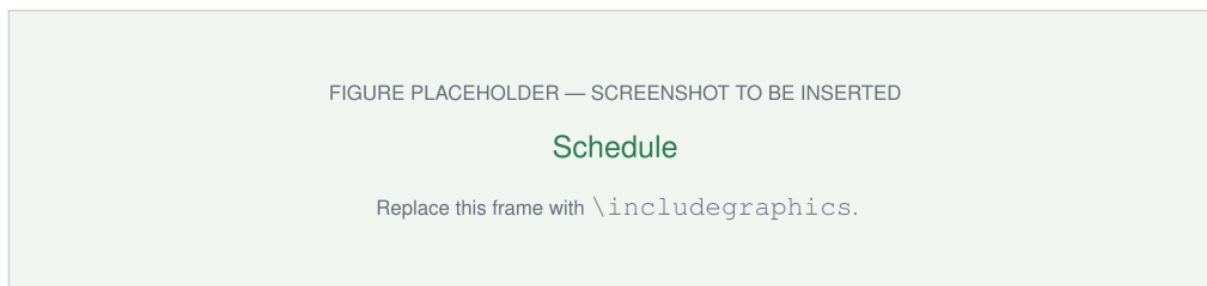


Figure 3: Location colour-coding, practitioner unavailability rendered as labelled bands, and imported external busy time shown as opaque blocks whose titles are visible only to their owner or on an explicitly shared calendar. Being a clinic administrator is deliberately not a third way to read them.

4.3 Patient communication

Goal. Hold the clinic's WhatsApp conversation with a patient inside the patient's record, rather than on one staff member's personal phone.

Behaviour. Inbound and outbound messages form one thread grouped by local calendar day, with automated reminders merged in as bodiless events. The composer resolves into exactly one of six states with a fixed precedence — no permission, no phone number, opted out, provider not configured, outside the 24-hour window, ready — and each state explains itself rather than presenting a disabled button. A conversation can be escalated as needing a clinician by a person; no keyword list and no model makes that judgement.

Engineering. WhatsApp's 24-hour customer-service window is computed from the patient's most recent inbound message and closes exactly at the boundary, pinned by a test. The client's answer is advisory: the server re-reads every input from the database and re-runs the same pure function before sending. Message bodies are encrypted at rest on the same terms as clinical notes, on the stated grounds that "the swelling is worse today" is exactly as sensitive as a note typed by the clinician.

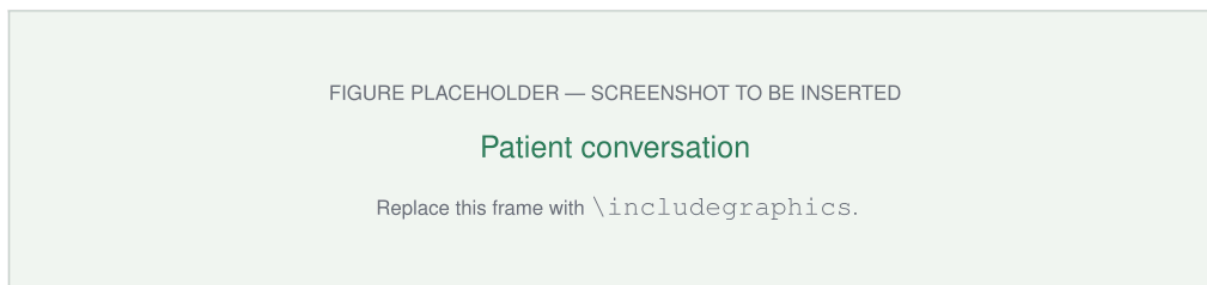


Figure 4: Per-message delivery state drawn from provider receipts, automated reminders shown as events with no body, and the composer rendered in its resolved state with the reason it cannot send. The five-word clinician-facing delivery vocabulary is deliberately distinct from the six-state provider vocabulary underneath it.

4.4 Automated appointment reminders

Goal. The day-before reminder happens without anybody remembering.

Behaviour. An hourly scheduled job selects appointments approximately 24 hours out, claims each one, resolves an approved provider template in the right language, and sends. Failures are classified into a closed 15-value vocabulary across four groups — recipient, configuration, provider and lifecycle. Exactly two of those fifteen are treated as transient and earn a bounded retry, three attempts with 5, 20 and 80 minute backoff. Consent is re-checked on every retry rather than inherited from the original attempt.

Engineering. The reminder row is created *before* the provider is called, under a unique constraint on `(appointmentId, channel)`, so a concurrent or repeated sweep loses the insert rather than sending twice. Retries reuse the same row through a conditional update rather than creating a second one. The candidate query spans a window wider than any single clinic’s “tomorrow”, because clinic offsets range from UTC−12 to UTC+14, and then filters per clinic.

4.5 Public self-service booking

Goal. Let a patient book without telephoning, without exposing anything the clinic did not choose to publish.

Behaviour. A clinic may publish a booking page, off by default. Enabling it is not sufficient: a location must carry parseable opening hours, a practitioner must be individually marked publicly bookable, a procedure must be individually marked publicly bookable, and a practitioner-location pair must exist. All five gates fail closed into one identical 404, so the page cannot be used to confirm that a given clinic is a Medix customer. The published practitioner and location lists are derived from the valid pairs rather than listed independently — publishing both lists separately published their cross product.

Engineering. Submissions carry a client-generated idempotency key, and every value is re-derived server-side: the clinic is re-resolved through the same gates, the service and practitioner are looked up in the arrays the resolver returned rather than queried by the submitted identifier, duration comes from the service record, and the submitted time must still appear in freshly computed availability. Minimum lead time defaults to two hours and the booking horizon to 30 days, both configurable per clinic. A booked slot is explicitly not a hold; the database owns the guarantee.

FIGURE PLACEHOLDER — SCREENSHOT TO BE INSERTED

Public booking

Replace this frame with `\includegraphics`.

Figure 5: The patient-facing page: opted-in services only, practitioner-location pairs, and slots computed as working hours minus unavailability, external busy time and existing appointments. Every refusal path — rate limit, invalid pair, taken slot — returns the same shape as a fully-booked clinic, so the form is not an oracle.

4.6 Guided clinic setup

Goal. Get a clinic from an empty workspace to something usable without a support call.

Behaviour. A sequential questionnaire derives clinic configuration — professional identity, specialties, workplaces, weekly hours, whether assistants will be invited, an initial service catalogue — and writes real domain rows: locations, procedure types, working hours. The progress counter is derived from which questions this identity will ever be asked, rather than from a hardcoded total. Professional identity is derived from the role the account already holds and is never submitted as a form field, which is what makes it unforgeable.

Engineering. The derivation is a synchronous pure function. It is entirely rule-based and performs no network call of any kind; the reasoning and the guard that enforces it are covered in Section 7, decision 9. A separate readiness summary counts actual domain rows rather than re-reading the answers, because the previous version described the onboarding JSON and could therefore tick a workplace that was never written.

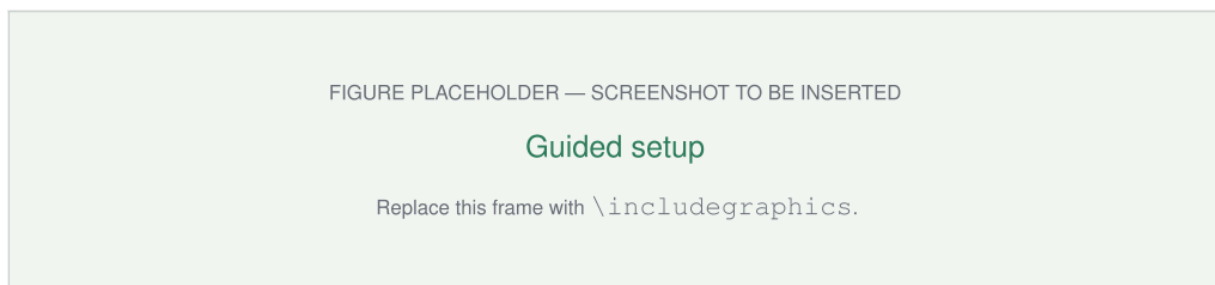


Figure 6: One question per screen with a derived progress count. Copy that promised questions about “practitioners, rooms and daily workflow” was removed because the flow asks about none of the three, and a test now asserts the subtitle does not name them — a small example of treating product copy as something that can be wrong rather than merely unpolished.

5 System Architecture

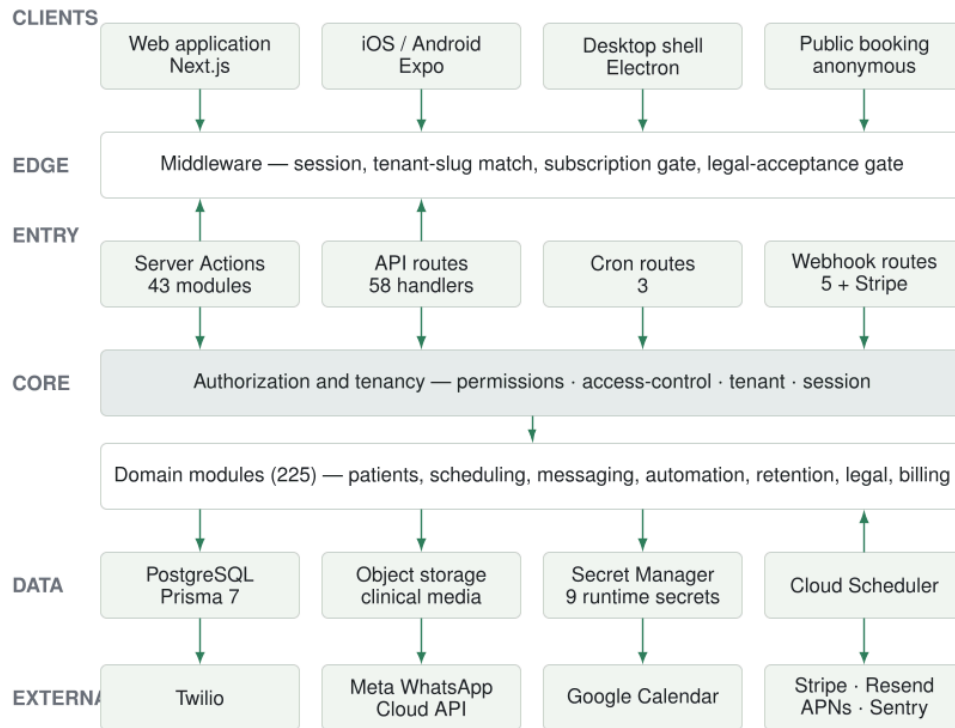


Figure 7: Request and integration topology. Every client, including the native applications and the anonymous booking page, converges on the same authorization and tenancy core; no client holds authorization logic of its own. Cron routes and webhook routes bypass the middleware and authenticate themselves, which is why they appear at the same layer as the API routes rather than beneath the edge.

Layer	As implemented
Frontend	Next.js App Router with React Server Components; Tailwind over a CSS-variable token foundation; Radix and Base UI primitives; Framer Motion; <code>react-big-calendar</code> ; Recharts; a command palette. Six locales with cookie-persisted selection, bidirectional isolation and right-to-left layout for Arabic.
Server	Server Actions for mutations, Route Handlers for APIs, webhooks and cron. Edge middleware for session presence, tenant-slug matching, subscription state and the legal-acceptance gate. Zod for input validation at every boundary.
Authorization	Two layers: a static role-to-permission map (52 permissions across 9 roles) and dynamic query scoping. A single tenant-resolution function is the chokepoint used by more than 170 call sites.
Data	PostgreSQL through Prisma 7 with a pluggable driver adapter (node-postgres or Neon serverless, selected by connection string). 56 models, 51 enums, 52 checksum-verified migrations.
Storage	A driver interface with three implementations — Google Cloud Storage, Cloudflare R2 and a local driver. The active backend is named by environment variable rather than detected, so which system holds clinical media is a written deployment decision. Uploads follow a quarantine-then-promote lifecycle with presigned URLs.
Messaging	A three-method provider interface implemented by a Twilio adapter and a Meta Cloud API adapter, with per-clinic resolution and an eight-value normalised error vocabulary.
Calendar	Three integrations: an outbound clinic mirror (write scope), an inbound personal busy-time import (read scope, push channels plus incremental cursors), and Apple EventKit on device. The first two are separate OAuth grants against separate tables.
Billing	One provider-neutral entitlement event model with adapters for Stripe, Apple StoreKit 2 and Google Play, idempotent by provider event identifier and ordered by provider timestamp.
Deployment	Container image on Google Cloud Run, deployed by immutable digest with zero traffic and promoted as a separate human decision. Cloud Build for staging, Workload Identity Federation for keyless CI authentication, Secret Manager for all nine runtime secrets, Cloud Scheduler for the three cron routes.

Table 2: Architecture summary. Every row corresponds to code or configuration present in the repository.

6 Data Model and Multi-Tenancy

Medix is multi-tenant by `Clinic`. Tenancy is path-based (`{clinicSlug}/...`) at the routing layer and column-based (`clinicId`) at the data layer. Neither of those is the enforcement that matters.

6.1 Principal entities

Entity	Role in the model
<code>Clinic</code>	The tenant, the account and the authorization boundary. Carries subscription state, default locale and timezone, and public-booking policy.
<code>User</code>	A staff member. Holds a tenant <code>role</code> and, separately and deliberately, a boolean <code>platformAdmin</code> for control-plane authority.
<code>Location</code>	A workplace owned by a clinic — hospital, branch, office, home-visit service. Not a second tenant.
<code>UserLocation</code>	Which practitioner works where, with an optional weekly pattern for that specific pairing. Its <i>absence</i> is a refusal.
<code>PatientLocation</code>	Additional site associations for a patient, alongside an optional primary location.
<code>Patient</code>	The person. Carries visibility (<code>clinicShared</code> , primary doctor), per-channel opt-out, a stated locale, and acquisition source.
<code>PatientAssignment</code>	The explicit assignment relation that patient visibility reads.
<code>Appointment</code>	An intention. Carries practitioner, location, procedure type, arrival timestamp, acquisition source and a public-booking idempotency key.
<code>Visit</code>	What actually happened. Snapshots the procedure name and recall interval as they stood at the time.
<code>ProcedureType</code>	The clinic's service catalogue: duration, decimal price, recall interval, follow-up template, and a <i>separate</i> opt-in for public bookability.
<code>PatientMessage</code>	One message in a conversation. Body encrypted; submission state and delivery state held as distinct columns.
<code>MessageDeliveryEvent</code>	Append-only ledger of what providers reported about a message.
<code>AppointmentReminder</code>	One automated send, with its own claim, retry counter, failure reason and delivery state.
<code>PatientFollowUp</code> / <code>FollowUpTemplate</code>	Structured post-procedure protocols with due dates, assignment and documented completion.
<code>PatientConsent</code>	Consent records across a five-value closed vocabulary, now constrained at the database level.
<code>AuditLog</code> / <code>SecurityEvent</code>	Append-only accountability records. <code>userId</code> is a plain column, not a foreign key.
<code>LegalHold</code> / <code>RetentionRun</code>	Preservation obligations, and the retention engine that must respect them.
<code>PrivilegedAccessSession</code>	A time-bound, scoped break-glass grant.

Table 3: 18 of the schema's 56 models.

6.2 The failure mode composite keys prevent

Consider a `Photo` row carrying both `clinicId` and `patientId`, each with its own single-column foreign key. Clinic A's identifier satisfies the foreign key to `Clinic`. Clinic B's patient identifier satisfies the foreign key to `Patient`. Both constraints pass. The database has no way to object, because *no constraint relates the two columns to each other*. The row is internally contradictory and perfectly valid.

That is not a hypothetical. The same shape appears on `PatientAssignment` — the table that patient-visibility checks read to decide who may open a record — and on `PatientConsent`, the legal record of what a patient agreed to. A cross-tenant row in either is an access-control answer, or a consent, attributed to the wrong clinic. Discovering it later means auditing which decisions were made on the strength of it.



Figure 8: The composite key shape. Because the child's single `clinicId` scalar participates simultaneously in its own clinic relation and in the composite relation to its tenant-owned parent, a row naming one clinic and another clinic's patient is not merely unchecked — it is unrepresentable.

6.3 How it is applied

There are **33 composite references**, originating in **24 models**, against **11** (`clinicId`, `id`) unique targets — `User`, `Patient`, `Location`, `Appointment`, `Photo`, `FollowUpTemplate`, `UserCalendarConnection`, `ExternalCalendar`, `ProcedureType`, `Visit` and `PatientMessage`.

`Visit` carries a second, wider target: (`clinicId`, `patientId`, `id`). Including the patient identifier in the key that consents and photographs reference is what makes attaching one patient's consent to another patient's visit impossible in PostgreSQL rather than merely checked in application code.

Two directional details are deliberate. `ON UPDATE` is `RESTRICT` on every composite key, because a cascading update on a tenant key would silently move rows between clinics — precisely the failure the constraint exists to prevent. `ON DELETE` preserves whatever the baseline established, so deleting a patient still removes their assignments and consent records.

6.4 How the conversion was de-risked

The conversion depended on an assumption nobody had verified: that Prisma would accept a model in which the same `clinicId` scalar is used by two relations at once — the child's own clinic relation and the composite relation to its parent. If it did not, every affected model would have had to drop its direct clinic relation and reach the clinic through the parent, changing query shapes across the application.

Rather than reading documentation, I settled it empirically in a throwaway spike: a three-line schema edit, a compile-time type probe using `@ts-expect-error` for the negative cases so a wrong assertion fails the build rather than passing silently, and the same probe compiled against the unmodified schema to measure which findings were actually *caused* by the change. The result was a two-error delta: the compound unique input is additive, so `findUnique({ where: { id } })` keeps working everywhere, and the single genuine narrowing had no call site in the repository. The spike also surfaced a real property of the accepted shape — the two foreign keys on `clinicId` carry disagreeing `ON UPDATE` rules, making a clinic-identifier rewrite fail. Clinic identifiers are never rewritten, so it is inert; it is recorded rather than left to be discovered later.

The migration sequence that followed is eleven files, ordered most-sensitive last, each stating its own locking behaviour, its transactional guarantee, and the pre-flight query that must return zero rows before it may run. A read-only verification script executes those same checks against a live database inside a `READ ONLY` transaction.

7 Key Engineering Decisions

Ten decisions, each stated as the problem, the decision, the reasoning, the trade-off accepted, and where the evidence lives.

7.1 Tenant consistency as composite foreign keys

Problem Independent single-column foreign keys can each be satisfied while together describing a cross-tenant relationship, as developed in Section 6.2. The affected tables include the one patient-visibility checks read and the one that records legal consent.

Decision Convert nine relation families to composite (`clinicId`, `parentId`) keys referencing (`clinicId`, `id`) unique targets, in eleven ordered migrations, with `ON UPDATE RESTRICT` and preserved delete behaviour.

Reasoning Tenant isolation enforced only by query discipline holds until the first query written without it, and that query's output looks entirely normal. A constraint converts a silent wrong answer into a refused write.

Trade-offs Every target table needs an additional unique index. Nested relation writes lose the ability to set `clinicId` explicitly. Each migration validates every existing row as the constraint is created, so one inconsistent row aborts the deployment — which is why the verification script runs first. The sequence is long and cannot be collapsed.

Evidence Eleven migrations dated 2026-08-05 to 2026-08-13; `docs/adr/ml-composite-fk-shape.md`; `scripts/verify-tenant-integrity.ts`; `tests/guards/verify-tenant-integrity.test.ts`; `tests/integration/tenant-scope.test.ts`.

7.2 Double-booking refused by the database

Problem Two concurrent booking requests can both read “this slot is free” and both write. The window between the read and the write is where the double booking is created, and no amount of application checking closes it: the check and the insert are separate statements, and a second transaction can land between them.

Decision A PostgreSQL GiST exclusion constraint over (`practitionerId WITH =, tsrange(date, date + durationMinutes) WITH &&`), predicated to exclude appointments with no practitioner and those in canceled or no_show status.

Reasoning An exclusion constraint evaluates at write time under the database's own concurrency control, so the second transaction blocks until the first resolves and is then refused. The range is half-open, so 10:00–10:30 and 10:30–11:00 are adjacent rather than overlapping — back-to-back appointments are the normal case and must stay legal. The status predicate means a cancellation frees its slot immediately without deleting the clinical record.

Trade-offs Requires the `btree_gist` extension, because the constraint mixes an equality operator on a scalar with an overlap operator on a range. Appointments with no practitioner fall outside it; the migration names this and describes itself as “a floor, not a ceiling”, identifying a practitioner picker in the booking dialogue as the precondition for full coverage. Recognising the violation at the application boundary required pinning Prisma 7's driver-adapter error shape by hand, because the documented `code` and `meta` fields are not populated — the `SQLSTATE` and constraint name arrive on the error's `cause`.

Evidence `20260814000000_appointment_no_double_booking`; `src/lib/scheduling-conflict.ts`; `tests/integration/double-booking.test.ts`, whose 13 cases include one asserting the constraint is installed as an exclusion constraint rather than an index or a trigger, and one asserting the extension is present.

7.3 A provider-neutral messaging abstraction

Problem Medix began on a single shared Twilio sender serving every clinic. Clinics need their own WhatsApp Business number, which means Meta’s Cloud API: a different transport, different template semantics, a different error vocabulary, and — critically — a different answer to the question of which clinic an inbound message belongs to.

Decision A three-method interface (`status`, `sendFreeForm`, `sendTemplate`) with two adapters, and a resolution function that prefers a clinic’s own connected Meta number and falls back to the shared sender. Vendor error codes are normalised into eight kinds. Templates are addressed semantically — by kind, channel, provider and locale — never by a vendor identifier at the call site.

Reasoning The domain should not know which vendor carried a message. The normalised error set is what allows one retry policy, one status vocabulary and one set of operational metrics to serve both providers. Resolving templates semantically is what allowed a single environment variable holding one template for one deployment to become a per-clinic, per-locale table.

Trade-offs A template reference means different things per provider — a Twilio Content SID versus a Meta registered template name — so the abstraction carries an opaque identifier plus a declared variable count rather than a body. The approved text lives at the provider, because the provider approved it and a second copy would be a second thing to keep in step with an approval Medix does not control. A null provider is a supported state the entire product must handle rather than an error.

Evidence `src/lib/messaging/ (10 modules); tests/integration/reminder-provider-routing.test.ts; tests/integration/message-template-resolution.test.ts; tests/guards/automation-boundary.test.ts` which pins that the retired Twilio-direct module stays deleted and that templates are requested semantically.

7.4 Reserve before send, and four independent idempotency keys

Problem Every messaging path can execute twice, for four unrelated reasons: a double click or a second browser tab; a retried HTTP request; a webhook redelivered because the provider did not see a 2xx; and two cron sweeps overlapping. Each has a different natural key, and a single shared mechanism would fit none of them.

Decision Four database-enforced keys rather than four remembered code paths. A unique `sendOperationId` on outbound messages, generated by the client and inserted *before* the provider is called. (`provider`, `providerMessageId`) unique on inbound messages. (`provider`, `providerMessageId`, `state`) unique on delivery receipts. (`appointmentId`, `channel`) unique on reminder claims.

Reasoning Ordering is the whole mechanism. Reserving the row first means a second submission loses the insert and never reaches the provider. Sending first and recording afterwards lets both requests through, and the duplicate is then something the patient has already received. Scoping the provider key per provider matters because two services can mint the same identifier string and that collision is not a duplicate.

Trade-offs A message can end in a terminal `unknown` state: the provider may have accepted it, but Medix could not persist the acknowledgement. That state is deliberately never retried automatically and surfaces to the user as “uncertain” rather than as success or failure. Detecting the constraint conflict required pinning Prisma’s PostgreSQL adapter contract, which preserves quotation marks around camel-cased column names in the error payload.

Evidence `src/lib/patient-messages.ts; src/lib/reminder-delivery.ts; src/lib/prisma-conflict.ts; tests/integration/outbound-message-send.test.ts; 20260910010000_public_booking_idempotency` whose commentary notes that the exclusion constraint already made a double submission harmless and that what was actually wrong was the *message* — “that time was just taken”, about a slot the visitor had taken themselves.

7.5 Submission state and delivery state as separate columns

Problem “Sent” conflates two different facts: what Medix did, and what the provider proved. A reminder can legitimately be `skipped` because the patient opted out — a state no provider was ever asked about. A receipt can arrive before its own send row has been written, out of order, or for a message identifier belonging to another deployment sharing the same provider account. All three are ordinary operation, not anomalies.

Decision Two columns with two vocabularies. A lifecycle status owned by Medix, and a delivery state set only from receipts and never from a send response. Receipts land in an append-only ledger, unique per `(provider, message, state)`, and are merged into the message row by confidence rank so that a late-arriving `sent` cannot downgrade a known `delivered`. Unmatched receipts are retained and reconciled on a later sweep. A third, five-word vocabulary translates both into something a clinician can act on.

Reasoning Collapsing them means either inventing a delivery state for a message nobody sent, or losing the reason it was not sent. Both destroy the answer to “why did this patient not get their reminder”, which is the question the whole subsystem exists to answer.

Trade-offs Three vocabularies to keep aligned; a guard test pins the rank array against the schema enum so they cannot drift. The ledger grows monotonically. Receipts carry no body, recipient or patient reference by design, which means diagnosing delivery never requires decrypting anything — but also means the ledger alone cannot answer content questions.

Evidence `src/lib/messaging/delivery.ts`; `src/lib/message-thread-view.ts`; `tests/integration/message-thread-view.ts`; `tests/guards/messaging-delivery-boundary.test.ts`.

7.6 Platform authority moved off the role enum

Problem `superadmin` was a value of the tenant role enum, which made platform authority and clinical authority mutually exclusive: every platform operator had no clinic, so one person could not both operate Medix and run their own practice without maintaining two accounts.

Decision A separate `User.platformAdmin` boolean, held independently of `role`, plus a per-session active mode so that a laptop switched to platform mode does not switch a phone. The authorization check reads the database row; a token claim is used only to *deny* tenant access, never to grant platform access.

Reasoning There is exactly one platform authority, so a boolean is the honest shape; inventing a second hierarchy nobody needs would be the same mistake in new clothing. Reading the row rather than the claim means a revoked operator loses access at the next request rather than at token expiry.

Trade-offs The change had a severe latent failure mode, and finding it is the part worth recording. Once a dual-mode identity’s role is `doctor`, every tenant-scoped read that did not pass through the route guard would have served that tenant’s clinical data while the session was operating the control plane. There were 111 such call sites, twelve of them API routes: patient lists, patient detail, exports, PDFs, photographs. Under the old model they were safe *by accident*, because a platform operator’s role was `superadmin` and the tenant resolver threw. The symptom that surfaced it was innocuous — `/api/patients` answering 200 [], empty only because that tenant happened to have no patients. The fix was to refuse inside `getClinicId`, the one function all 111 already called, rather than to patch call sites, because a boundary that wide only stays closed if it is closed in one place.

Evidence `src/lib/tenant.ts`, whose comment records the incident; `src/lib/identity/session-mode.ts`; `20260921000000_dual_mode_identity`; `tests/guards/dual-mode-identity.test.ts`; `docs/adr/phase-`

7.7 A dead helper removed rather than corrected

Problem The tenant module contained an `assertClinicAccess` helper whose second branch read, unconditionally, “if the role is `superadmin`, return”. Any call site that adopted it as *the* tenant check would have handed a platform account every row it guarded, silently, with no audit trail and no second factor.

Decision Delete it. It had zero callers in the entire tree, which is the only reason it was never a live vulnerability.

Reasoning A corrected version would still be dead code inside the module that *defines* the tenant boundary, and dead code in that module is precisely the condition that let it survive unread. It also contradicted the function eleven lines above it, which throws for exactly the same input: one module cannot hold two answers to the same question.

Trade-offs A future caller genuinely needing “may this session reach this clinic’s resource” must now compose it from the tenant resolver and a comparison against the resource’s own clinic identifier — slightly more verbose at the call site, and correct by construction because the resolver refuses a platform account.

Evidence `src/lib/tenant.ts`, which retains the removal and its reasoning as a comment so the shape cannot return by copy-paste.

7.8 Derive what can be derived; snapshot what must not move

Problem Two opposite failure modes, both silent. A stored copy of a derivable fact goes stale when its inputs change. A live reference to an editable fact rewrites history when that fact is edited.

Decision Clinical recalls, front-desk arrival state and the patient journey are derived at read time and have no tables at all. `Visit`, conversely, stores a snapshot of the procedure name and the recall interval as they stood when the visit was recorded.

Reasoning A recall is a question the existing rows already answer: the visit knows when it happened, the procedure knows its interval, the schedule knows whether the patient has since booked. A recall table would be a fourth copy of those three, and a rescheduled visit, a corrected interval or a cancelled booking would each leave a stale row behind with no signal. The inverse holds for the visit: editing a procedure’s recall interval today would otherwise rewrite the due date of every visit ever recorded against it, so a patient who was due in six months becomes overdue by four and nothing in the record shows that anything changed.

Trade-offs Derived views cost computation on every read and cannot record facts the source tables do not hold — “this patient was contacted, replied, and asked not to be” would require the table that deliberately does not exist. The recalls module says so explicitly: persisting earns its place when there is something new to remember, and until then it does not. Snapshot columns are null on rows written before they existed and fall back to the live value, which is the behaviour those rows already had.

Evidence `src/lib/recalls.ts`; `src/lib/front-desk.ts`; `src/lib/patient-journey.ts`; `Visit.procedure` and `Visit.recallIntervalDaysSnapshot` in the schema.

7.9 Removing the model call from onboarding, and guarding its return

Problem An earlier version of clinic setup posted the entire answer set to a third-party language model and asked it to invent a clinic configuration. The branch was unreachable in practice — an earlier return covered the only value the relevant field could take — but the environment key was documented in the deployment guide as an optional feature that produced “richer config”.

Decision Delete the path; delete the key from every example file and deployment document; and pin the removal with a test asserting that the module contains no `fetch` call of any kind and that the derivation function does not return a `Promise`.

Reasoning The reachability was incidental. The first person to add a second setup path would have silently begun sending clinic configuration — specialties, workplaces, staffing, working patterns — to a third party, because the documentation told them the key was supported. Deterministic derivation is also simply the better fit for this workflow: the output is a typed configuration object validated against a schema, the input is a closed questionnaire, the result must be reproducible and explainable to the clinic that will live with it, and the whole thing must work offline in a deployment that has no outbound network allowance. A non-deterministic generator would have added a dependency, a failure mode, a latency budget and a data-processing disclosure in exchange for variability nobody asked for.

Trade-offs Setup cannot accommodate a clinic that does not fit the professional catalogue; such a clinic completes configuration in Settings afterwards. The absence of a `Promise` return type is itself part of the guarantee, since an asynchronous signature would leave room for a network call to return without anything else changing.

Evidence `tests/guards/onboarding-boundaries.test.ts`; no language-model SDK appears in any package manifest in the repository, verified across the web, mobile and desktop manifests.

7.10 Deploy by immutable digest, with no traffic

Problem Deploying by image tag means the running revision can change underneath you when the tag is re-pushed, so “what is in production” has no stable answer. Shipping a build directly to full traffic makes a bad release a production incident rather than a discarded candidate. And running migrations inside the container means that, with scale-to-zero, several cold-starting instances race each other to apply the same migration.

Decision The release workflow resolves the immutable image digest, deploys *that digest* with `-no-traffic` and a revision URL tag, smoke-tests the candidate at its own URL, counts its error-level logs, and stops. Promotion to serving traffic is a separate, human decision. Migrations never run in the container; they are an operator step with their own gated workflow.

Reasoning A digest names one specific set of bytes. A candidate serving zero traffic can be exercised as thoroughly as you like without anybody noticing. Separating migration from deployment removes the race entirely rather than mitigating it with a lock.

Trade-offs Two-step releases are slower and require the candidate to be tracked as an artefact. Cloud Run refuses `-no-traffic` when creating a service’s first revision, so that one case is branched explicitly and warns. `-set-secrets` and `-set-env-vars` *replace* rather than merge, so a single stale workflow line can strip configuration — which is why a guard test asserts that the release workflow and the credential-attachment workflow name identical sets.

Evidence `.github/workflows/deploy-production.yml`; `Dockerfile`, which states “NO MIGRATIONS RUN HERE” and gives the scale-to-zero reasoning; `tests/guards/production-runtime-configuration.test.ts`; `tests/guards/production-release-order.test.ts`.

8 Reliability, Privacy and Safety Engineering

On compliance language

Medix makes **no compliance certification claim**. It is not HIPAA certified, not GDPR certified, not ISO certified, not a regulated medical device, and holds no regulatory approval. The repository's own privacy documentation is explicit that the service agreement, data-processing agreement and privacy policy require legal review before publication, and that the currently published privacy policy is a placeholder. What follows is a description of implemented controls, in engineering terms. Where a control is designed against a specific obligation, that is stated as design intent, not as compliance.

Authorization. Two layers: a static role-to-permission map (52 permissions across 9 effective roles, with one legacy role aliased) and dynamic query scoping. Several permissions are split where a single capability would merge two decisions that belong to different people. Placing a legal hold and releasing one are separate capabilities, on the recorded grounds that the person who records “litigation, case reference X” and the person who decides the litigation is over need not be the same person. Recording the data-transfer consent for a transfer the holder themselves initiated is separate from the general power to attest consent against a patient, so a visiting consultant can complete a transfer without acquiring the broader capability.

Tenant isolation. Composite foreign keys (Section 6); a single tenant-resolution chokepoint that refuses platform sessions; a route-level slug comparison against a database-fresh value re-read on every request rather than a stale token claim; and a removed helper that would have granted platform accounts everything it guarded.

Encryption. AES-256-GCM with a random 96-bit initialisation vector, an authentication tag, and a versioned `enc:v1`: envelope, applied to clinical notes, medical history, message bodies, held-message bodies, imported calendar event titles, TOTP secrets and provider access tokens. A missing or malformed key causes new clinical writes to *fail* rather than fall back to plaintext. A start-up probe verifies that the key currently held can still read data already stored, so a key rotation that would render the corpus unreadable is detected at boot rather than at the first patient record somebody opens.

Per-channel opt-out, with precedence. Opt-out is a boolean per channel rather than one global switch, because a password reset the patient requested, a data export they are entitled to, and a breach notification a regulator requires are not automation anybody may decline — a single switch that silently swallowed them would be a compliance failure dressed as a courtesy. Keyword detection matches a whole message, not a sentence containing the word, and runs *before* routing, so an ambiguous or unroutable STOP still takes effect across every patient record holding that number. The opt-out check sits above the provider and window checks in the composer's precedence specifically so no later branch can turn an approved template into a way to reach somebody who asked not to be reached, and it is re-evaluated on every retry rather than inherited.

Audit trail. Append-only by application discipline, proven by a guard that reads every file under `src/` and fails on any update, delete or upsert against the audit table; the one deliberate exception, a redaction script, lives outside `src/` precisely so it cannot be reached from a request. `AuditLog.userId` is a plain column rather than a foreign key, because it used to cascade from the user table — and the accounts most worth investigating are the ones somebody might want removed. Four account- administration actions record evidence *before* acting and refuse outright if that write fails, on the reasoning that administering somebody else's account is only defensible because it was recorded, so the record is a precondition rather than a description. Ordinary audit logging remains best-effort by contrast, deliberately: it is a receipt, and a logging outage must not become a clinical outage.

Deletion lifecycle. No one-click delete exists. The sequence is request, with a categorised and length-capped reason; approval, which starts a 30-day recovery window; and a separate manual purge confirmation. The reason field is capped at 280 characters as a control rather than a formatting preference — the unbounded textarea it replaced invited a narrative, and a narrative about a patient is where clinical detail arrives. Legal holds can block a purge. Staff account deletion is a tombstone rather than a row removal: credentials are destroyed and the email address released, while the name survives as professional attribution on clinical records, because several cascades that are correct for live state sit next to two that are accountability history.

Break-glass access. Platform operators hold no clinical access by default; the constant that would represent it reads `not_active`, the console renders that state at all times, and refusal notices appear where the data would otherwise be. Where privileged access exists it is time-bound (30 minutes by default, 240 maximum), read-only, scoped to one tenant and one patient, requires one of five recorded reasons and one of six named capabilities, and writes its evidence first. The policy module imports no Prisma, no session and no request, so every branch is reachable from a plain object and “a session scoped to one patient refuses a different patient” is a two-line test rather than a seeded clinic.

Secrets. Nine runtime secrets are supplied by Secret Manager and referenced by name in the deployment. CI authenticates through Workload Identity Federation with no stored key. Build arguments carry non-secret values only, because a build argument is recoverable from image history. A repository-hygiene guard asserts no environment file is tracked, using a narrow pathspec so the query is proportional to the answer rather than to the repository.

Transport and browser surface. Strict headers on every route: `X-Frame-Options: DENY`, `X-Content-Type-Options: nosniff`, a referrer policy, a permissions policy disabling camera, microphone and geolocation, HSTS with a one-year max-age and subdomains, and a Content Security Policy with `frame-ancestors 'none'`, `object-src 'none'`, `base-uri 'self'`, `form-action 'self'` and an explicitly empty remote image allow-list. The policy still permits inline and evaluated script, which the configuration acknowledges in place rather than leaving implicit.

Sessions and credentials. Web sessions expire after 8 hours. Native bearer tokens are HMAC-signed, compared in constant time, bound to a server-side session registry, and expire on a 30-day idle and 90-day absolute ceiling, with the row stamped revoked so the security page stops listing it. Failed sign-ins lock an account for 15 minutes. TOTP is RFC 6238 with the last accepted timestep recorded, because a one-time password is one-time only if something remembers it was used.

Observability without patient data. Two separate channels: a closed-vocabulary operational ledger for outcomes the product predicted, and an exceptional channel for things that threw, with a pinned caller list of exactly two. The monitoring SDK is initialised blind — no breadcrumbs, no console capture, no request bodies, no default personal data — because the repository has roughly forty raw console call sites and the SDK’s default console capture would have shipped every one of them as telemetry the moment a DSN was configured. Ten behavioural tests build real reports and assert that patient values are absent. Expected refusals are deliberately excluded from the paging channel, on the reasoning that a clinical product refuses things constantly and correctly, and a paging channel that includes those stops meaning anything within a day.

Backup and recovery. Backup, restore, wrong-key and storage-reconciliation drills run in CI on every build. The restore verification tests four properties, and tests the schema property by *attempting* an overlapping insert and requiring refusal rather than by reading the system catalogue, because a constraint can exist and be `NOT VALID`. A separate drill provisions a disposable database and two roles, applies the real evidence-protection SQL with only role names substituted, asserts each write that must succeed and each that must fail, then simulates an ownership-stripping restore and asserts the protection is gone, re-applies it, and asserts it is back.

FIGURE PLACEHOLDER — SCREENSHOT TO BE INSERTED

Privacy Centre

Replace this frame with `\includegraphics`.

Figure 9: Consent coverage, retention policy, legal holds, deletion requests and audit access in one clinic-facing surface. The trust statements distinguish *configured* controls from *enforced* ones, and the module that generates them imports nothing, so the mapping from mechanism to claim exists in exactly one place and a new receipt kind must pick an existing mechanism rather than inventing a claim.

9 Testing and Verification Strategy

9.1 Shape of the suite

Layer	Files	Declared	What it exercises
Architectural guards	111	1,272	Source, stylesheet, locale and workflow invariants. No database, no build, no browser.
Integration	97	1,199	Real PostgreSQL, single fork, no file parallelism, every table truncated between tests.
Unit and domain	78	≈775	Pure modules: availability, slot generation, composer state, delivery merge, retention, entitlement, locale resolution.
Native	30	303	Expo client logic and server contract parity, in Node rather than on a device.
End-to-end (Playwright)	8	50	A production build plus Chromium, including accessibility scanning.
Total	324	≈3,599	In 958 <code>describe</code> blocks; 71,996 lines of test code against 88,546 lines of application code.

Table 4: Measured on 14 September 2026. See the epistemic note below on what “declared” means.

What these numbers are, and what they are not

File counts are exact, obtained by file-system enumeration.

Declared-case counts are static: they count `it(` and `test(` expressions in the source with one consistent anchored pattern. They are a *lower bound* on executed assertions, because parameterised constructs such as `it.each` and the end-to-end surface and persona matrices expand into many cases at run time. The 50 declared end-to-end cases, for instance, expand across a surface matrix, a persona matrix and an accessibility matrix.

Runtime-expanded counts are not mine. The project’s own status record dated 10 September 2026 reports the expanded run as 2,641 unit and native, 1,227 integration, 315 browser tests and 271 accessibility scans. I did not execute the suite for this document and therefore present those figures as an attributed status record rather than as an independent measurement.

9.2 The architectural guard layer

This is the layer most likely to be unfamiliar, and it is a genuine differentiator. 111 test files read the repository as *data* and assert structural properties. They enforce rules that cannot be expressed as types and that a code review will eventually miss.

Category	Representative invariants
Tenancy and authorization	No unsafe query pattern reaches a tenant-scoped table; no control-plane query opens a clinical row; no alternate route bypasses a permission check; cron routes authorize.
Patient data and privacy	No audit row is ever mutated anywhere under <code>src/</code> ; no patient value appears in a monitoring report; no cache header permits storing a clinical response; no accessible name leaks patient data.
Database and migrations	Every migration's checksum matches; the migration policy holds; the consent-type constraint exists; the tenant-integrity script covers the relations it claims.
Secrets and environment	No environment file is tracked; the CI encryption key is quoted; the environment fails closed; no credential is published.
Design system and accessibility	Contrast recomputed from the token file; dialogue and overlay contracts; form composition; motion contract; navigation architecture; font preload budget.
Internationalisation	Key-set and interpolation-placeholder parity across six locales; plural handling; bidirectional isolation; script typography.
Messaging and automation	Delivery rank matches the schema enum; no recall or follow-up can reach a sender; templates are addressed semantically; email delivery claims nothing the record cannot support.
Release posture	The release workflow's environment set matches the credential-attachment workflow's; the production target is what it claims; release ordering holds.
Meta-tests on the suite	The integration harness's own refusal rules are tested; the end-to-end surface matrix is compared against every page file on disk, so a shrinking matrix is caught rather than silently passing.

Table 5: The guard layer, by category. Several of these files state in their own headers that they are defence in depth rather than proof, and name the integration test that proves the behaviour — an epistemic distinction I consider part of the design.

Two implementation notes indicate the maturity of the approach. One guard replaced a regular expression with a brace-walking analyser, on the recorded reasoning that a check which is both false-positive prone and blind to most of the tree “produces confidence rather than coverage”. Another enumerates sources through `git ls-files` and separates server modules from client components before applying its rule, because the vulnerability class it targets is a confused-deputy problem and the two contexts are not equivalent.

9.3 The integration harness refuses to run in the wrong place

The integration suite requires a real PostgreSQL database, which creates an obvious hazard: pointing it at the wrong one truncates every table. Three gates decide whether it may start.

1. A dedicated test database URL must be set.
2. It must differ from the application's own database URL, compared against both the process environment and the environment file.
3. After connecting, the *server's* reported `current_database()` must appear in a six-name allow-list, and the host must not match a managed-provider pattern covering Neon, Supabase, Amazon RDS and Azure.

The third gate is the one that matters, and the harness says so: it reads the server rather than trusting a string, so a connection string that lies is caught. Truncation is ordered by table name, after unordered truncation produced lock-order deadlocks that surfaced as unrelated foreign-key

failures in other files.

The integration configuration is deliberately *not* a merge of the base configuration. Merging concatenated the include arrays and produced a run that executed the entire unit suite, no integration test at all, and still reported a green summary — the failure mode that makes a test suite worse than none.

9.4 End-to-end and accessibility

Playwright runs against a real production build on a dedicated database, with one worker and no parallelism. The single-worker constraint is recorded rather than assumed: a parallel run produced 76 failures whose messages all read like genuine authorization results, and every one was a worker reseeding the database underneath another. The end-to-end environment deliberately points storage at a bucket that does not exist, because the local driver is refused in a production build and the test needs the production code path.

10 Continuous Integration and Release Discipline

10.1 The integration pipeline

`ci.yml` runs on every push to the canonical branch and every pull request, as two jobs in parallel, with in-progress runs cancelled on the same reference.

Native job — no database, no simulator. Install both lockfiles, type-check the mobile application against two TypeScript projects, lint it, run the native suite, then run `expo prebuild` and assert properties of the *generated* Android manifest: that two sensitive permissions are absent from the release manifest, that a third is explicitly removed, and that it is still present in the debug manifest. Asserting against generated output rather than against the source configuration is the point — whichever manifest is on disk is the one that ships.

Build job — against a PostgreSQL service container, in order:

1. An *inverted* pre-flight assertion: the schema verifier must exit non-zero against an empty database. A verifier that cannot detect an empty database as drifted cannot detect anything.
2. `prisma migrate deploy`, then `migrate status`, then schema verification — which must now pass.
3. Type-check and lint.
4. The unit and guard suites.
5. Create a dedicated integration database, migrate it, run the integration suite against it.
6. Production build.
7. Create a third database, migrate it, install Chromium, run the end-to-end suite.
8. **Recovery drill**: create two further databases, seed synthetic data under one encryption key, `pg_dump` and `pg_restore` without owners or privileges, and verify the restore.
9. **Wrong-key drill**: take a checksum of the encrypted columns, attempt an update under a different key, require it to throw, and require the ciphertext to be byte-identical afterwards.
10. Storage reconciliation, report-only, since CI has no bucket.
11. On failure, upload the Playwright report with a seven-day retention.

The CI encryption key is quoted in the workflow, and a guard test pins the quoting. Unquoted, YAML parsed it as the integer zero, which made the encryption helper a pass-through — so every CI run had been exercising the application with encryption at rest silently disabled.

10.2 Release and operations workflows

Nineteen workflows. Three are automatic: continuous integration, staging deployment on a staging branch, and an audit workflow on a production branch. The remaining sixteen are manual-dispatch only, and nearly all require a typed confirmation string — `deploy production`, `migrate production`, `provision production`, `prove roles`, `publish legal`, `move tag` — so that no production action is one mis-click away.

The production release workflow gates in this order before it authenticates to the cloud at all: type-check, lint, the full guard suite, migration-integrity verification, legal-readiness verification. Only then does it authenticate, and it then verifies that every referenced secret already exists by name and that the token it holds actually reaches the expected project. It resolves the image digest, deploys with no traffic, smoke-tests the candidate at its tagged URL, counts error-level logs attributable to that specific revision, and reports.

10.3 Migration safety

Three independent mechanisms, because migrations are where a mistake is least reversible.

Checksum manifest. 52 entries, one per migration directory, SHA-256 over line-ending-normalised bytes. Editing a migration that has already been applied fails the check in CI rather than aborting a deployment halfway. The mechanism exists because one migration was added in one commit and edited in another, which is exactly the condition that makes `prisma migrate deploy` abort.

Target fingerprinting. Before acting on a production database URL, a verifier parses it through a long list of structural checks — scheme, credentials present, no fragment, no malformed percent-encoding, hostname shape, no connection pooler, port range, database-name pattern, exactly one SSL mode, no unexpected query parameters — and then compares a SHA-256 of host, port and database name against an expected fingerprint in constant time.

Classified drift. The schema verifier derives its empty-or-not verdict from Prisma's own `migrate diff -exit-code` rather than from parsing, prints the raw diff unconditionally, and fails closed when the classifier and the exit code disagree. It exists in that shape because the original naive implementation split the diff on semicolons and discarded every statement preceded by a comment: a 30-kilobyte diff parsed as zero statements.

10.4 Deployment topology

A three-stage Debian-based container image — Debian rather than Alpine because of the image-processing library's glibc prebuilt binaries. The runner copies only the standalone output, runs as a non-root user, and starts the server directly. A placeholder database URL is required at build time because the Prisma client is constructed at module import.

Staging builds through Cloud Build rather than a direct image push, because the direct path accepts no build arguments. Staging deliberately omits the monitoring ingest origin, so staging client monitoring is blocked by the Content Security Policy rather than quietly reporting into the production project. Staging cancels in-progress runs because it is a mirror; production never cancels, because it is a record.

11 Selected Technical Challenges

11.1 Inbound WhatsApp routing on a shared sender

What made it hard. With one Twilio number serving every clinic, an inbound message names a phone number and nothing else. If two clinics each have a patient with that number, routing it to either one discloses a patient's message to the wrong clinic — a disclosure the rest of the product exists to prevent, arriving through the one door that has no authenticated caller.

How it is addressed. The system refuses to guess. A message is routed only when *both* the clinic and the patient resolve uniquely. Otherwise it becomes a held message: a row deliberately carrying no clinic identifier and no relation to a clinic or a patient, because giving it a tenant would mean choosing one. It retains the candidate clinic identifiers for the human who will resolve it, keeps the body encrypted, and is visible only to a platform operator. Filing it copies ciphertext across without decrypting it and re-checks that the patient still holds the originating number. The reason is a closed two-value vocabulary — no patient match, or ambiguous patient match — so the two situations remain distinguishable in operations.

Phone matching itself is a raw SQL sweep on the last nine digits, backed by a functional index, followed by an exact re-check in application code using the same normaliser the send path uses. The ambiguity disappears entirely once a clinic connects its own Meta number: the webhook then names the receiving number, and the owning clinic is *known* rather than inferred.

11.2 Time, in a product that spans time zones

What made it hard. Working hours are wall-clock and belong to a location. Appointments are stored without a time zone. A reminder must go out “the day before” in the clinic's own calendar. Daylight-saving transitions make some wall-clock times non-existent and others ambiguous. And a single cron sweep has to serve clinics whose local day starts across a 26-hour spread.

How it is addressed. The reminder sweep queries a window deliberately wider than any single clinic's tomorrow, because offsets run from UTC−12 to UTC+14, and then filters per clinic with a shared helper. Slot generation resolves each candidate wall-clock time in the clinic's zone and *drops* those that do not exist or are ambiguous rather than resolving them to a guess. All-day calendar events are anchored by resolving the offset for that specific date, not by assuming a fixed offset for the zone. The slot grid steps by the service duration rather than by a fixed interval, so a 45-minute procedure tiles correctly instead of being offered on 15-minute boundaries it cannot fit.

11.3 Making automation honest

What made it hard. The interesting states are all between success and failure. A reminder can be skipped because the patient opted out. A message can be accepted by a provider whose acknowledgement could not be persisted. A receipt can arrive before its own send row exists, out of order, or for a message identifier belonging to another deployment sharing the provider account. None of these is an anomaly; all are ordinary operation.

How it is addressed. Each is modelled separately rather than collapsed: a lifecycle status, a submission state including a terminal `unknown`, an append-only receipt ledger merged by confidence rank, a bounded retry policy that treats only two of fifteen failure reasons as transient, and a distinct clinician-facing vocabulary of five words. Unmatched receipts are kept and reconciled on a later sweep within a bounded window rather than discarded. The failure-reason vocabulary is closed rather than free text specifically so that causes can be counted, filtered and acted on, and so that a provider's own sentence — which carries account identifiers — never reaches a row a settings screen renders.

11.4 Two-way calendar synchronisation without a mirror loop

What made it hard. Medix writes appointments into a clinic's Google calendar and separately reads a practitioner's personal calendar for busy time. Naively, every mirrored appointment returns as busy time blocking itself, and the practitioner's diary fills with phantom conflicts. Separately, Google's push channels expire and notifications are best-effort, so a lapsed channel produces a calendar that looks connected and is silently stale.

How it is addressed. The outbound writer stamps every event with a private extended property naming the Medix appointment. The inbound importer resolves that marker first, falls back to a clinic-scoped lookup on the stored event identifier, and records the result; the availability predicate then requires the mirror reference to be null. A 30-minute cron sweep re-synchronises stale calendars and renews channels approaching expiry, registering the new channel *before* stopping the old one. The product's freshness vocabulary deliberately never contains the word "live".

The two integrations are separate grants against separate tables for a reason stated in the schema: the clinic's outbound mirror is workspace-owned, but a practitioner's personal calendar carries details belonging to that person and not to their employer, so a clinic-level row would let whoever administers the clinic inherit the connection and its private event titles. Privacy is enforced in the server projection rather than the browser: when a title may not be read, the key is *omitted from the payload entirely* rather than present and hidden, so it cannot travel in the serialised component payload. Being a clinic administrator is explicitly not a third way to read it.

11.5 Webhook trust, four different ways

What made it hard. Six externally-triggered endpoints, each from a different provider with a different verification scheme, all unauthenticated in the ordinary sense, all subject to replay, and all of which must decide which tenant a request concerns without believing anything the request says about it.

How it is addressed. Twilio requests are verified by request signature over a reconstructed signed URL. Meta requests are verified by HMAC-SHA256 over the exact raw bytes — the body is read once as text, because the signature covers bytes and not a re-serialised object — compared in constant time. Google Calendar push notifications are matched against a stored SHA-256 of the channel token, never the token itself, with the resource identifier also required to match so a notification for one calendar cannot be replayed against another. Stripe events are deduplicated by making Stripe's own event identifier the primary key of a table, so the database refuses the second delivery rather than the application remembering to.

In every case tenant identity comes from a persisted row, never from the request, and the response is chosen to stop a provider retrying something that will never become valid: a missing signing secret answers 503 and pages a human, an unverifiable request answers 403 or a benign 200 depending on whether retrying could ever help, and a database error answers 500 so the provider retries — which is safe precisely because of the unique constraints. A provider that is switched off in a given deployment answers 404: the route does not exist rather than existing and refusing.

12 Product and User-Experience Decisions

Duplicate detection reports; it never blocks and never merges. Twins share a surname and a birthday; a family shares a phone line. Candidates are surfaced with masked contact hints and the person decides. The check is also tenant-scoped absolutely, because surfacing a match from another clinic would disclose that a person is a patient somewhere else.

Import never overwrites. There is no update path in the module at all, for the reason given in Section 4.1. Planning and applying are separate functions and the plan is pure, so the preview a person approves is the same object the apply step consumes — there is no second code path whose behaviour could differ from the one that was reviewed, which is the usual way a dry run comes to mean nothing.

The composer has states, not a disabled button. Six named states with a fixed precedence, each explaining itself. A disabled control with no explanation is indistinguishable from a broken one, and in a messaging surface the difference between “you may not” and “you cannot yet” matters to whoever is holding the phone.

Automated messages appear in the conversation, without a body. Making them visible prevents the clinic from being surprised by a message it did not know had gone out. Showing them without content prevents the conversation from looking like something a person wrote. Attribution is structural rather than a flag: automated sends and human sends live in different tables, so a missing author name means the staff account was deleted, not that the system sent it.

Self-service WhatsApp setup that reports what actually happened. The connection flow exchanges the code, upgrades to a long-lived token, verifies the number and subscribes the webhook. Two of those steps were previously missing and both failed silently: a short-lived token works during setup and is dead by morning, and without the subscription call the connection looks perfect and receives nothing forever. A failed subscription no longer fails the connection — it is recorded as unproven, surfaces as “configuration incomplete”, and a health probe retries it. The connection row separately records when the credential was last *proven* to work, as distinct from when it was last written down.

Recalls are surfaced without becoming outreach. The system computes who is overdue and shows it to staff as a clinical work list. It is wired to no sender. The decision record gives the reasoning: a recall date is a clinical fact, sending it is a commercial act, the two are indistinguishable in a database, and the distance between them is where a clinical product becomes a marketing one. A guard test keeps the work lists free of any path to a sender.

Public booking is opt-in at every level. Clinic, location hours, practitioner, procedure and pair. A clinic’s catalogue holds revision surgery and biopsy review alongside a consultation, and enabling public booking must not put all of them on the internet.

Nothing infers a patient’s language. Preferred locale is null until stated, falling through to the clinic’s own language. It is initialised from the language a patient chose while booking themselves, because that is a person stating a preference in their own request rather than a system inferring one from a phone country code or a name — guesses that would be wrong for exactly the patients least able to complain about it.

Setup claims only what it asks. Copy describing configuration the flow does not collect was removed, and invented capacity figures presented as though the clinic had stated them were removed with it. A test now asserts the remaining copy does not name the three things it used to promise.

One design system, dark by default, enforced. Every value defined once, with a guard that recomputes contrast ratios from the token file. A redundant selector arm was removed after it caused a real diagnosis cost: two mechanisms with different specificity meant a root carrying one and contradicting the other rendered the wrong theme with no error anywhere.

Six locales including right-to-left. Roughly 2,900 keys per locale with parity, plural and bidirectional-isolation guards, Gregorian date formatting pinned so one clinic never sees two numbering systems, and a specific guard for the handful of strings that frame first-run setup — because a parity check proves keys exist, not that anybody translated them, and four locales shipped English on the first and last screens a new customer ever sees.

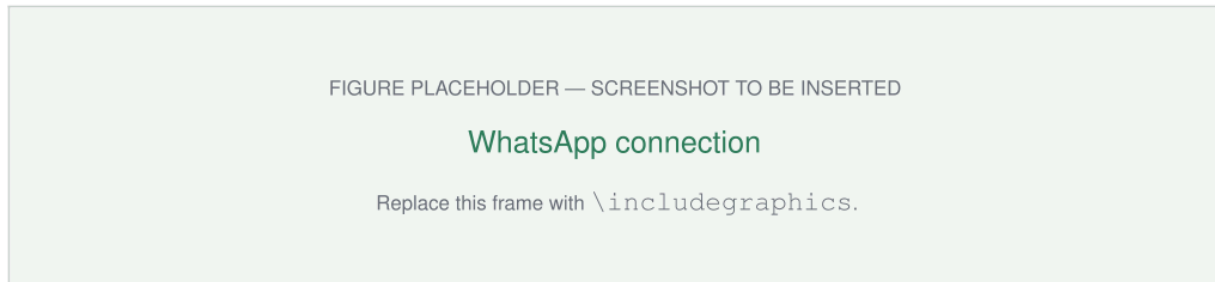


Figure 10: Connection state, verified display number, webhook-subscription status, credential expiry warning, and a health probe distinguishing “what this row claims” from “when that claim was last tested”. A token revoked at the provider would otherwise leave the status reading connected forever, and the product would discover it when a patient did not get a reminder.

13 Current Status: Implemented, Partial, Absent

The three tables below are the authoritative statement of what exists. Nothing elsewhere in this document contradicts them; where this document and these tables disagree, these tables are correct.

13.1 Implemented and verified

13.2 Partial or in progress

13.3 Explicitly not present

14 Roadmap

Every item below is **future work**, drawn from the repository’s own deferred lists, decision records and stated gaps. None is partially shipped.

1. **Apply database-level audit protection.** The two-role grant script exists, is re-runnable, and has been rehearsed against a disposable database including a restore that strips it. Applying it converts an application-enforced property into a database-enforced one and changes what the trust surface is entitled to claim.
2. **Close the double-booking coverage gap.** Appointments without a practitioner sit outside the exclusion constraint; the migration identifies a practitioner picker in the booking dialogue as the precondition for full coverage.
3. **Give the outbound clinic calendar connection a status column,** so an expired token is distinguishable from never having connected — the same deficiency the messaging connection model already fixed, named in the schema.
4. **Store distribution.** Signing and upload, App Store Connect records, in-app purchase products, screenshots from a built binary, a Firebase project for Android push, and a decision on localising the native client.
5. **Measure a real recovery.** The drill runs on every build; what is missing is an actual recovery-time and recovery-point measurement against the deployed database, and a decision on cross-region

IMPLEMENTED

Deployed environment	Container on Google Cloud Run, served at <code>medix.healthcare</code> ; responded HTTP 200 on 14 September 2026, including the public account-deletion and security resources. Exercised with synthetic data only — see Section 1.5.
Core clinical domain	Patients, appointments, visits, procedure catalogue, locations, follow-ups, recalls, consents, photographs, exports.
Authorization	9 roles, 52 permissions, three visibility modes, dual-mode platform identity, time-bound and patient-scoped break-glass access.
Tenancy	33 composite foreign keys from 24 models onto 11 targets, a read-only verification script, integration and guard coverage.
Scheduling	Exclusion-constraint double-booking prevention, practitioner unavailability, per-location working hours, bounded and paged range queries.
Messaging	Twilio and Meta adapters, per-clinic resolution, inbound routing with held-message resolution, 24-hour window, opt-out precedence, append-only delivery ledger.
Automation	WhatsApp and email appointment reminders with claims, semantic template resolution, bounded retry and receipt reconciliation.
Calendar	Outbound clinic mirror, inbound personal busy import with incremental cursors and push channels, 30-minute freshness sweep, device-side EventKit.
Public booking	End-to-end with five fail-closed gates, idempotency and patient matching that is not an oracle.
Billing model	One convergent entitlement event model across Stripe, Apple and Google; idempotent, order-aware, cross-clinic replay refused.
Platform control plane	Proven clinical-access boundary, account administration with evidence written before the act, selected-tenant context that is a lens over metadata rather than a sign-in.
Privacy machinery	Consent records with a database-enforced type constraint, legal holds, retention evaluation, gated deletion lifecycle, deletion receipts, versioned and hashed legal documents with an in-product acceptance gate.
Internationalisation	Six locales, $\approx 2,900$ keys each, RTL Arabic, with parity and typography guards.
Quality infrastructure	324 test files, CI including backup/restore and wrong-key recovery drills, 19 gated workflows, 52 checksum-verified migrations.

PARTIAL

iOS and Android clients	Code-complete against the same server: 16 screen and layout files, 14 dedicated native API routes, contract-parity tests, prebuild and an unsigned release build verified. Not distributed — not on the App Store, not on Google Play, not in TestFlight. Signing, store records and store products remain outstanding. The native client currently ships no localisation.
Push notifications	Transport implemented and tested; the Apple key is bound from Secret Manager and the release refuses to deploy without it. Nothing calls the sender yet — a product decision, not a credential gap. Android push additionally requires a Firebase project that does not exist.
Desktop client	An Electron main process, preload script, icon set and <code>electron-builder</code> configuration exist — two TypeScript source files in total. No released artefact is evidenced in the repository.
Database-level audit immutability	Append-only is enforced by application discipline and a guard test. The SQL granting the two-role separation that would enforce it in PostgreSQL is written and re-runnable and has not been applied to the deployed database. The product's own trust surface reports the control as <i>configured</i> rather than <i>enforced</i> for this reason.
Production restore	Backup, restore, wrong-key and reconciliation drills run in CI on every build. No production restore has ever been performed , no recovery-time or recovery-point figure has been measured against the deployed database, and there is no cross-region replication.
Web billing (Stripe)	Implemented and deliberately switched off . No Stripe secret exists in the deployed secret store; checkout and the customer portal refuse with 503 after authentication; the billing page states that online billing is not enabled. Enabling it is a commercial decision plus an external dependency.
Legal documents	Versioned, content-hashed and gated in-product, with publication and verification workflows. The published privacy policy is a placeholder that states of itself that it governs nobody. A reviewed policy must be published as a new version directory, never as an edit, or every recorded acceptance is invalidated.
Monitoring	17 alert policies and 14 log-based metrics defined as code with an idempotent apply script. The repository's own monitoring README is internally inconsistent about whether they are currently instantiated in the cloud project — its header says they are live, its closing line says nothing exists — and I could not resolve this from the repository.
Tenant lifecycle	A clinic can be created and not retired. There is no lifecycle column and no operator action. Recorded as a known gap rather than invented; three empty test tenants exist and cannot currently be cleanly retired.
Performance	A measured font-weight reduction; paint metrics unmeasured, because the build environment's browser pane is hidden and a hidden page never paints. An earlier alarming figure was identified as an artefact of exactly that and discarded rather than chased.
Android build	Prebuild succeeds and the generated manifest is asserted in CI. No Gradle build has been performed — an environment gap, not a code one.

NOT PRESENT

Artificial intelligence	Medix contains no AI or language-model functionality. No such SDK appears in the web, mobile or desktop manifests. The one historical model call was removed from onboarding and a guard test prevents its return. Medix should not be described as an AI-powered product.
Real clinical usage	No real patient records exist. All data is synthetic. No pilot clinic has been onboarded; the pilot topology is a written design.
Automated recall or marketing outreach	Recalls and follow-ups are computed and displayed to staff and are wired to no sender, by recorded decision and enforced by test.
Media messaging	Inbound non-text WhatsApp messages are counted and not stored. The reasoning — download, encryption at rest, retention classification, privileged-access path — is written out in the adapter.
Reply templates outside the 24-hour window	Only the appointment reminder template is resolved; there is genuinely nothing to send outside the window, and the composer says so rather than failing at the provider.
Service worker / full PWA install	The application is installable and serves a manifest, maskable icon and per-theme colours. A service worker is a considered refusal : caching authenticated responses would place patient data in on-disk cache storage outside every deletion path, and caching the shell would serve a stale build of clinical software with no signal.
Platform impersonation	There is no path by which a platform operator becomes a clinic user. The guard suite asserts its absence.
Compliance certification	None claimed, none held.

replication.

6. **Tenant lifecycle.** A reversible, auditable suspension state and a retirement path, with the retention and legal-hold interactions decided rather than assumed.
7. **Publish reviewed legal documents** as new versioned entries, never as edits to existing ones.
8. **Wire the push sender.** The transport exists and is tested; what remains is deciding which events justify interrupting a clinician.
9. **Deferred by decision record:** team-based patient sharing as the evolution of the per-patient shared flag; doctor-initiated archive requests; realtime synchronisation; and a service worker, which remains a refusal rather than an unfinished item and whose bar for revisiting is written down.

15 What I Learned

A guarantee belongs at the lowest layer capable of expressing it. I began by enforcing tenant separation in queries and reviewing carefully. That works until the first query written without it, and the output of that query looks entirely normal. Moving the rule into composite foreign keys, an exclusion constraint and unique indexes changed the failure mode from “wrong data, silently” to “the write is refused”. Everything above the constraint then becomes a better error message rather than the guarantee itself, which is a much easier thing to get right.

The hard part of an integration is the ambiguous middle, not the call. Sending a WhatsApp message is one HTTP request. Knowing what happened to it is a submission state, a delivery ledger, a reconciliation sweep for receipts that arrive before their sends, a bounded retry policy over a closed failure vocabulary, and a terminal “uncertain” state that must never be retried. I underestimated that ratio badly at the start, and it is the single strongest predictor I now have of how long an integration will actually take.

Deleting code can be the architectural decision. The onboarding model call, the unconditional

platform grant in the tenant module, the per-clinic accent colour, the parallel theme selector: each was removed rather than corrected, and each removal is pinned by a test. Writing down *why* something is absent has stopped me from re-adding it more than once, and it converts a piece of tribal knowledge into something the repository enforces on its own.

Structural tests enforce boundaries a type system cannot express. “No control-plane query opens a clinical row” is not a type. “No audit row is ever mutated anywhere under `src/`” is not a type. These are the rules most likely to be broken by somebody acting reasonably in a file far from where the rule was decided, and a test that reads the source tree is the cheapest mechanism I have found for holding them.

A product surface has to be able to say “I don’t know”. Reporting the audit trail as configured rather than enforced is less impressive and correct. So is telling a clinic that a message’s delivery is uncertain rather than guessing. A clinician who cannot distinguish “nothing is wrong” from “I am not permitted to look” will assume the first, and designing for that distinction changed several screens and one database column.

Choosing not to automate something is product work. The data to automate recalls has existed for months. Not wiring it, writing down why, and enforcing the absence with a test was more valuable than the feature would have been — and articulating the difference between a clinical fact and a commercial act was the part that took the longest.

Operational tooling is part of the software. Fingerprint-checked migration targets, typed confirmation strings, deploying by digest with no traffic, a test harness that refuses to run against a managed host, an inverted CI assertion that proves the verifier can detect drift at all: none of it is visible to a user, and all of it is what makes changing a clinical system at eleven at night a defensible thing to do.

Reading your own failures carefully is a technique. A large share of the strongest mechanisms in this codebase exist because something went wrong first — an unquoted CI variable that silently disabled encryption, a parallel test run whose 76 failures all looked like real authorization results, a diff parser that read 30 kilobytes as zero statements, an empty API response that was empty only by luck. The mechanism was never the interesting part; the five minutes spent asking what class of mistake it belonged to was.

16 Technology Stack

Listed for completeness. The architecture and the decisions in Section 7 are the substance of this case study; the library list is not.

17 Evidence and Verification

The repository is private. Everything below can be demonstrated in a walkthrough or screen share without exposing credentials, infrastructure detail or data.

Git history and authorship. 642 commits on the current line, 764 across all references, one author, 16 June to 12 September 2026. Authorship is verifiable with `git shortlog -sne` on both the current line and all references. Commit messages are statements of intent rather than change logs, and the development arc runs through twelve numbered phases visible in the history.

Architecture decision records. Six records covering the composite foreign key shape, the control-plane boundary, break-glass privileged access, the automated-outreach boundary, the scheduling

Category	Present in the repository
Language	TypeScript 5, React 19
Frontend	Next.js 16 (App Router, React Server Components), Tailwind CSS 3, Base UI and Radix primitives, Framer Motion, <code>react-big-calendar</code> , Recharts, <code>cmdk</code> , Sonner, <code>next-themes</code> , <code>react-hook-form</code> , Zod resolvers
Design system	Custom CSS-variable token foundation across nine stylesheets, light and dark themes, motion tokens, RTL support; guard-enforced
Backend	Next.js Server Actions and Route Handlers, Edge middleware, Zod validation, React Email, <code>@react-pdf/renderer</code>
Database	PostgreSQL, Prisma 7 with pluggable driver adapters (<code>node-postgres</code> , Neon serverless); 56 models, 51 enums, 52 migrations
Authentication	NextAuth JWT sessions backed by a server-side session registry, <code>bcrypt</code> , RFC 6238 TOTP with replay protection, recovery codes, step-up re-authentication, HMAC bearer tokens for native clients
Integrations	Twilio (WhatsApp), Meta WhatsApp Cloud API, Google Calendar API, Stripe, Apple StoreKit 2 and App Store Server Notifications v2, Google Play real-time developer notifications, Resend, APNs
Storage	Google Cloud Storage, Cloudflare R2 via the AWS S3 SDK, local driver; <code>sharp</code> for image processing
Mobile	Expo 52 / React Native 0.76, Expo Router, Secure Store, Local Authentication, Calendar, Contacts, Notifications, <code>react-native-iap</code>
Desktop	Electron 34, <code>electron-builder</code> , <code>electron-updater</code>
Testing	Vitest 3 across three configurations, Playwright 1.6, <code>axe-core</code> accessibility scanning
Infrastructure	Google Cloud Run, Cloud Build, Artifact Registry, Secret Manager, Cloud Scheduler, Workload Identity Federation, Cloud Monitoring, Docker
Tooling	GitHub Actions (19 workflows), ESLint 10 with custom architectural rules restricting storage-key propagation, advisory-lock usage and direct storage imports, <code>tsx</code> , Prisma CLI, Sentry SDK

and location foundation, and the identity roadmap. Each records what was rejected and why, which is the part that is difficult to reconstruct after the fact.

Schema and migrations. 56 models and 52 migrations with a checksum manifest. The migration files carry their own reasoning — locking behaviour, transactional guarantees, the pre-flight query that must return zero, and in several cases the production incident that motivated them.

Tests. 324 files. The guard and unit layers run in seconds with no database and are the fastest thing to demonstrate; the integration layer runs against PostgreSQL; the end-to-end layer runs against a production build.

Continuous integration. A single workflow run demonstrates the complete pipeline, including the backup/restore and wrong-key recovery drills and the inverted pre-flight assertion.

Deployment configuration. Cloud Run service configuration, keyless federated CI authentication, gated release and migration workflows with typed confirmations, and 17 monitoring policies defined as code.

The running application. medix.healthcare, along with its public account-deletion and security resources. A guided demonstration can be given on the synthetic demonstration clinic, covering the full arc: create a patient, detect a near-duplicate, schedule against a practitioner who is unavailable, watch the database refuse an overlapping booking, exchange a WhatsApp message, and inspect the delivery ledger.

Not disclosed

No connection strings, secrets, API keys, tokens, internal hostnames, bucket names, patient data or personal phone numbers appear in this document or are shown in any demonstration. The demonstration workspace contains synthetic records only.

A Measurement Methodology

Every quantity in this document was measured on 14 September 2026 against a clean working tree at commit `aafcea14`. The methods are listed so that any figure can be reproduced or challenged.

Quantity	How it was obtained
Commits, authorship, dates	<code>git rev-list -count, git shortlog -sne</code> and <code>git log</code> on both HEAD and <code>-all</code> .
Lines of code	<code>git ls-files</code> filtered by extension, piped to <code>wc -l</code> . The <code>src/</code> figure excludes the generated Prisma client, which is not tracked.
Models, enums, migrations	Pattern counts over <code>prisma/schema.prisma</code> and a directory listing of <code>prisma/migrations</code> , cross-checked against the checksum manifest entry count.
Composite foreign keys	Count of <code>references: [clinicId, id]</code> in the schema (33), of models containing at least one (24), and of <code>@@unique([clinicId, id])</code> declarations (11).
Permissions and roles	Count of entries between the array delimiters in the permission constant (52) and of members in the role enum (10, of which one is a legacy alias mapped to another).
Routes, actions, components, modules	<code>find</code> by filename convention under the relevant directories.
Test files	<code>find</code> for <code>*.test.ts</code> , <code>*.test.tsx</code> and <code>*.spec.ts</code> under <code>tests/</code> , per directory.
Declared test cases	A single anchored pattern matching <code>it (</code> and <code>test (</code> expressions, including modifier suffixes, applied uniformly across directories. This under-counts parameterised tests and is therefore a lower bound.
Locale keys	Recursive leaf count over each locale's JSON file.
Workflows, monitoring policies, metrics	Directory listings and a key count over the metrics definition file.
Deployed endpoint status	<code>curl -I</code> against the public origin and two public resources; response codes only, no body retained.
Runtime-expanded test counts	Not measured. Quoted from the project's own status record dated 10 September 2026 and attributed as such.
Monitoring deployment state	Not determined. The repository's monitoring README contradicts itself and the discrepancy is disclosed rather than resolved.

Where a figure could not be established reliably from the repository, this document uses qualitative language or states that the quantity is unknown. No number in this document was estimated.