Callmata

Engineering Design Doc

A multi-tenant, browser-based video-calling platform built for the least technical person in the family — one shared production instance where any family can self-register, get its own private room, and ring each other with a tap. No accounts, no app store, no links to fumble.

Brian Foster · **callmata.com**

August 2026 · v0.5.1 — A personal / portfolio project, built end-to-end; not a commercial product.

SECTION 1

What it is

Callmata exists because video calling is still too hard for the people who need it most. FaceTime needs the right device, Zoom needs links and accounts, WhatsApp needs contacts set up correctly — and a grandparent needs none of that to go wrong at 7pm on a Sunday. Callmata reduces the entire experience to three taps: open the icon on your home screen, tap your name, tap the person you want. They get a real ring — a push notification that behaves like an incoming call *and* a plain text message saying who wants to chat — and one tap later everyone is in the family's private video room.

What started as a hard-coded app for one family of five is now a **multi-tenant platform**: a single Vercel deployment where any family can register itself with a phone number, receive its own join code and auto-provisioned video room, and manage its own members — while remaining completely invisible to every other family on the instance. The design goal throughout is a specific kind of simplicity: **every hard thing moves to the server or a service, so the phone in an eighty-year-old's hand has nothing to configure.**

3	1	2	26	0
TAPS FROM HOME SCREEN TO RINGING	SHARED INSTANCE, MANY FAMILIES	RING CHANNELS — PUSH + SMS	API ROUTES	ACCOUNTS, PASSWORDS, OR APP INSTALLS

SECTION 2

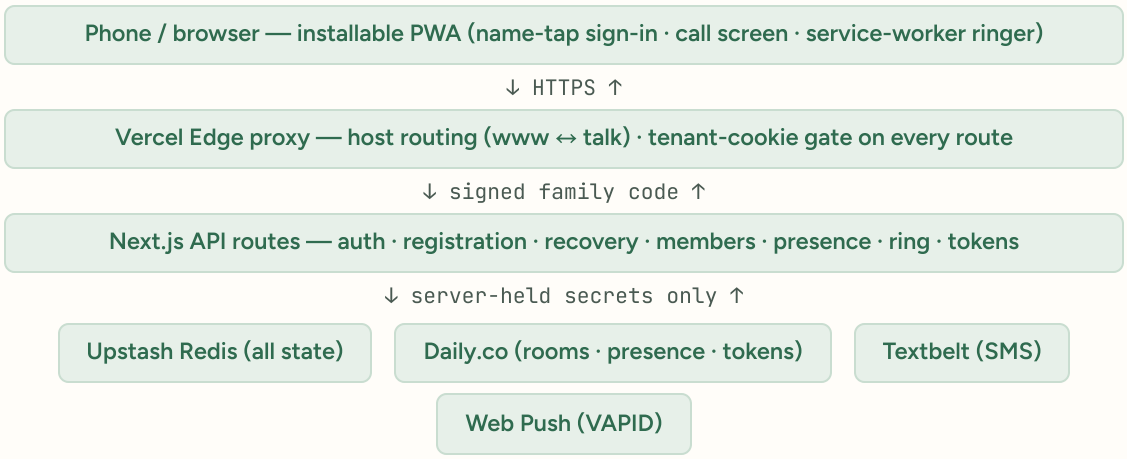
Technology stack

LAYER	TECHNOLOGY	ROLE & RATIONALE
Framework	Next.js 16 (App Router), React 19, TypeScript	One codebase serves the marketing site, the app, and all 26 API routes. Server components keep secrets and PINs off the client by construction.
Video / audio	Daily Prebuilt (`@daily-co/daily-js`)	Embedded call UI. Signaling, ICE/TURN, and mobile WebRTC quirks stay Daily's problem (Section 6). One private room per family, provisioned automatically at registration.
Persistence	Upstash Redis (REST)	The single backend for everything: family records, sessions, push subscriptions, rate-limit counters. Falls back to local JSON files in dev — same interface, zero setup.
Auth tokens	`jose` (HS256 JWTs)	Sessions, tenant cookies, and OTP-carrying tokens — all signed with one server secret. Chosen because it runs on the Edge runtime, where the route-gating proxy lives.
Ringing	`web-push` (VAPID) + SMS	Dual-channel call alerts. Push is the rich channel; SMS is the reliable one (Section 4).
SMS	Textbelt	The only provider, deliberately — it requires no 10DLC business registration for a personal-scale app (Section 7).
Abuse defense	Cloudflare Turnstile, Redis rate limiting	CAPTCHA plus 5/hour/IP throttling on the two public SMS-triggering endpoints — the cost of being one shared instance (Section 8).
Hosting	Vercel (serverless + edge)	Push-to-deploy; an Edge middleware does host-based routing (marketing on `www`, app on `talk`) and gates every route behind a family code.

SECTION 3

System architecture

One repository, one deploy target, two hostnames. An Edge-runtime proxy in front of every request does double duty: it routes `www.callmata.com` to the marketing page and `talk.callmata.com` to the app, and it refuses to serve anything beyond the front door unless the request carries a valid signed family-code cookie. The browser never holds a secret — the Daily API key, SMS credentials, and every PIN live only on the server.



Every per-family record in Redis is keyed by that family's code – the code is the tenant boundary (Section 5).

SECTION 4

Anatomy of one call

The core loop of the product is Grandma tapping "Call Brian." Everything in the system exists to make the next fifteen seconds work:

1. **Sign-in, without typing.** The home screen shows the family's members as big tappable avatars. Tap your name, enter a 4–6 digit PIN on an oversized on-screen keypad (no text field, no autocorrect, no keyboard switching). The server checks the PIN and issues a 30-day session token — sign-in is a rare event, not a daily chore.
2. **The ring.** Tapping a family member fires one request to `/api/push/notify`. The server sends a web-push notification and an SMS *in parallel* — it never waits on one channel to try the other.
3. **The push behaves like an incoming call.** The service worker shows a persistent notification (`requireInteraction`), vibrates in a ring pattern, and offers a "Join call" action. Tapping deep-links straight into `/call?from=Brian`.
4. **The SMS is deliberately plain.** "Callmata: Brian wants to chat." No link — the app is already installed and signed in on the recipient's phone, so the text only needs to say who's calling (and Textbelt blocks URLs by default anyway).

5. **Joining is server-brokered.** The callee's device asks `/api/token` for a short-lived Daily meeting token, minted server-side with the member's real name. The family's room is private — without a minted token there is no way in.

6. **Both land in the same room.** Daily Prebuilt renders the call. Joining is idempotent, so racing signals (Section 4a) can't double-join.

> **DESIGN NOTE — WHY TWO RING CHANNELS**
>
> Real-device testing settled this: **Android web push is not reliable enough to be "someone is calling."** OEM battery managers delay or silently drop notifications, and a missed ring is the one failure this product cannot absorb. So the roles are inverted from what you'd expect — **SMS is the required channel, push is the free bonus.** Both fire in parallel via `Promise.allSettled`; if one channel fails, the other has already left the building.

4a · The service worker as a telephone bell

Web push was designed for "your order shipped," not for ringing phones, and two workarounds make it behave. First, on tap the worker both navigates the existing app window *and* posts it an `open-url` message — because `client.navigate()` is unreliable inside iOS standalone PWAs. Whichever signal lands first wins; joining is idempotent, so the race is harmless by construction. Second, families share devices — a tablet on the kitchen counter is signed in as whoever touched it last. When a device subscribes to push for one member, the store **detaches that device from every other member first**, so the shared tablet rings for its current owner instead of for the whole family at once.

SECTION 5

Multi-tenancy — the family code is the identity

There are no user accounts anywhere in the system. The unit of identity is the **family**, and the family's public join code — six characters from an alphabet with no 0/O/1/I/L to confuse anyone reading it aloud over the phone — is simultaneously the thing you tell your relatives, the storage key, and the namespace for everything the family owns:

REDIS KEY	WHAT IT HOLDS
`family:<CODE>`	The family record: display name, private Daily room URL, members (`{name, phone, pin}`), and the head-of-household phone — set once at registration, never reassigned by the app.
`member-by-phone:<phone>`	Reverse index powering "forgot your code" recovery — maintained on write only, never on the hot read path.
`active-sessions:<CODE>`	Who is signed in on which device, per family.
`push-subscriptions:<CODE>`	Push endpoints per member, multi-device, with the shared-device deduplication from Section 4a.
`register-start:<ip>` · `recover-start:<ip>`	Rate-limit counters for the two public endpoints (Section 8).

Access is **two gates deep**. The Edge proxy admits nothing without a signed tenant cookie proving you entered a valid family code — before that, the app effectively does not exist. Past the gate, you still are nobody until you tap your name and enter your PIN, which yields a session token bound to that member *and* that family. Because every stored blob is namespaced by code, two families can each have a "Brian" without ever colliding.

> **FIELD TEST — THE BUG THAT WAS WAITING FOR FAMILY #2**
>
> With one family, a session token didn't need to say which family it belonged to — there was only one. The moment multi-tenancy shipped, that became a live cross-tenant hole: a valid session from family A presented alongside family B's tenant cookie would have been honored. The fix bakes the family code into the session JWT, and the server **rejects** a mismatch rather than "healing" it to the current tenant. Legacy tokens without a code are honored only for the original env-var family. The lesson: the single-tenant → multi-tenant transition doesn't add security requirements so much as **activate ones that were silently absent all along.**

Migrating family #1 without a migration

The original family — five members, configured entirely in environment variables — never went through `/register`. Rather than a one-off migration script, the store **seeds the legacy family into real storage the first time anyone reads it**, then serves it like any other tenant. The same pattern repeats across the codebase: head-of-household is backfilled on read for pre-feature records, and an older push-subscription shape is upgraded in place. Zero migrations have ever been run; every schema change is absorbed at the read path.

> **THE TRADEOFF WE ACCEPTED — KNOWINGLY**
>
> The original multi-tenancy plan was **one Vercel deployment per family** — perfect isolation, no shared abuse surface, and each family bringing its own API keys. It was rejected for the operational reality: one shared instance means one project, one Redis, one set of keys to rotate, and a registration flow a non-technical family can actually complete. The price is a real public abuse surface that a private family app never had — which is exactly what Section 8's guardrails were built to pay. The open question ("is public self-serve registration the right long-term call?") is recorded in the plan as deliberately unresolved — settled pragmatically, flagged for revisit if the app outgrows friends-and-family scale.

SECTION 6

Buy the hard part — Daily Prebuilt over raw WebRTC

The most consequential decision in the codebase is a thing it *doesn't* contain: there is no signaling server, no STUN/TURN configuration, no SDP negotiation, no per-browser WebRTC workaround matrix. Each family's calls run in a private room on Daily.co, embedded via Daily Prebuilt. For a solo-maintained app whose users cannot file useful bug reports ("the video is black again"), renting the battle-tested media path is the entire ballgame — signaling, ICE/TURN, mobile quirks, and call UI stay Daily's problem, true for one family or a hundred.

What the app keeps for itself is **control of the door**:

- **Rooms are provisioned server-side at registration.** Completing `/register` creates a private Daily room named for the family. If the Daily API key is missing, registration fails loudly — the system refuses to create a family with a broken room.

- **Tokens are minted server-side, per join.** The Daily API key never reaches the browser. A member gets a 2-hour meeting token carrying their real name; without one, the private room is a wall.
- **The lobby is removed.** Daily's "Are you ready to join?" prejoin screen is one more confusing step for an elderly user, so the app patches it off via Daily's REST API — with an in-process cache so each room is patched at most once.

> **FIELD TEST — REACT 19 DOUBLE-MOUNTS VS. THE SINGLETON IFRAME**
>
> Daily Prebuilt is an iframe that must exist exactly once, and React 19 Strict Mode mounts components twice in development — a combination that produced ghost call frames. The fix serializes all teardown through a module-level promise: any mount first awaits the destruction of any existing frame before creating its own. A cousin of the same class of problem: iPhone Safari silently refuses WebRTC on non-HTTPS LAN addresses during device testing, so the app detects a non-secure context and explains it in words a developer can act on — a real mobile-dev footgun converted into an error message.

SECTION 7

The SMS layer — one provider, one personal-scale constraint

SMS carries the product's most important message (the ring) and its most sensitive ones (OTP codes, PIN resets, member invites). The constraint shaping the whole layer is regulatory, not technical: US carriers now require **10DLC business registration** to send application SMS at scale — paperwork and monthly fees that make no sense for a family app. That disqualified Twilio and Telnyx, and made Textbelt — a paid reseller aimed at personal-scale use, no registration required — the provider. Singular:

THE SMS LAYER, COMPLETE

```
# One provider, one function — the whole layer.
function sendSms(to, message):
    if Textbelt is configured: send via Textbelt
    # dev with no provider: log the code to console instead — testable offline
    else: fail loudly
```

The single-function seam is deliberate: if Textbelt ever disappears, wiring in a replacement is a small, contained change. Textbelt's one real limitation is that it **blocks URLs in messages by default** — an anti-spam measure that would hurt most products, which lean on texted links for sign-in flows and deep-linking back into the app. Here it costs nothing, because the product never needs a link to get someone anywhere: the app is already installed and signed in on the recipient's phone, so a call alert's entire payload is "Brian wants to chat," and verification codes are six digits you type, not links you follow. The absence of links is arguably a feature — the messages look exactly *unlike* phishing, which is the right shape for texts aimed at the family members most targeted by scams. And **OTP codes are never stored anywhere**: the code is hashed with the server secret into a short-lived signed token that rides the user's own cookie, so verification is stateless and there is no OTP table to leak.

SECTION 8

Security & the abuse surface

Going from private app to public platform created exactly two dangerous endpoints: `/register/start` and `/recover/start`, both of which send SMS to an attacker-supplied phone number. Unguarded, that is a free SMS cannon and a registry-probing oracle. Four defenses stack on both:

1. **Cloudflare Turnstile**, verified server-side — and *fail-loud*: in production these endpoints refuse to run at all if the Turnstile secret is unset, so the protection cannot be silently misconfigured away.
2. **Rate limiting**, 5 attempts/hour/IP, fixed-window in Redis. The counter uses `INCR` -returns-1 to set its TTL exactly once — atomic by construction, no read-modify-write race across serverless instances.
3. **Anti-enumeration on recovery.** "Forgot your code" returns the identical response whether or not the phone matches a known family — only a real match actually sends an OTP. The endpoint cannot be used to probe who's registered.
4. **An admin SMS tripwire.** When either rate limit trips, the operator gets a text — throttled to once per hour per reason, and guaranteed never to throw into the request path. On an app with no monitoring stack, the alert channel is the same SMS layer the product already trusts.

The sign-in surfaces are throttled too. PIN login locks after ten *failed* attempts per 15 minutes per IP and family — only failures count, so a household sharing one Wi-Fi connection never notices it exists — and tripping it fires the same admin tripwire. All three OTP-verify endpoints cap code guessing at ten attempts per ten minutes, closing the brute-force window on both the 4–6 digit PINs and the 6-digit texted codes.

Inside the walls, authorization is deliberately coarse but human-shaped: any signed-in member can add a member or change their own PIN, but resetting *someone else's* PIN or re-sending an invite belongs to the **head of household** — the phone that registered the family, set once and never reassigned by the app. Phones and PINs never leave the server; the members API returns names only.

CREDENTIAL	LIFETIME	SCOPE
Tenant cookie	90 days	Proves the family code was entered; required by the Edge gate for every route.
Member session	30 days	Bound to one member *and* one family code; mismatches rejected, not healed.
OTP / login-link tokens	10 minutes	Carry a hashed one-time code; nothing stored server-side.
Daily meeting token	2 hours	Minted per join; the only way into a family's private room.

One gap is accepted on the record: rate limiting is per-IP only, with no global cap on families created. That's a fine posture for an unadvertised app — and it is written down in the plan as the thing to revisit first if that changes, which is itself the security posture: **known gaps documented beat unknown gaps assumed away.**

SECTION 9

Designing for grandparents

The visual layer — warm cream paper over deep greens, a serif brand face, a subtle paper-grain texture — is deliberately un-app-like: it reads calm, not techy. But the real design work is in what users never have to do:

NO TYPING, NO JARGON

Sign-in is tap-your-name (big colored avatar initials) plus a PIN on a custom oversized keypad. The app remembers the last user and greets them by name. Raw WebRTC and device errors are translated into plain sentences — "your camera is being used by another app" — before anyone sees them.

INSTALLED ONCE, BY A RELATIVE

Callmata is a PWA. The install prompt detects iPhone vs. Android — including iPads that report themselves as Macs — and shows platform-specific "Add to Home Screen" steps, on the assumption that the tech-comfortable family member does this step once during a visit. After that it's just an icon.

RECOVERY THAT MATCHES HOW FAMILIES WORK

Nobody remembers a code they entered once, two years ago, on a phone a grandchild set up. Recovery is by phone number: prove you own the head-of-household phone via OTP, get the code back, PIN reset included. The forgotten-credential path is a first-class flow, not a support email.

ESCAPE HATCHES FOR HUMANS

"Not your family?" backs out of a wrong code. "Open the room without ringing" lets you wait in the call quietly. Presence on the home screen shows who's in the call and whose alerts are on — polled every few seconds, so tapping "Call" is never a shot in the dark.

SECTION 10

Operations, cost & what's next

BUILT FOR SERVERLESS REALITY

Every function invocation may land on a fresh instance, so nothing load-bearing lives in process memory. Sessions survive instance churn: a valid session cookie can restore its server-side record — but only for an actual member of the family, never as a way in. Redis no-ops gracefully in local dev; the same code runs against JSON files on disk.

COST PROFILE

The stack is assembled from free and near-free tiers: Vercel hosting, Upstash Redis, Daily's free video tier, and pennies per SMS through Textbelt. There is no monthly bill worth mentioning — a structural requirement for software meant to run quietly for years for a handful of families.

OBSERVABILITY, PERSONAL-SCALE

A live dev panel (secret-gated in production) shows who's in the call, the family roster with per-member status, and who's waiting — the tool the notify flows were designed against. Abuse pages the operator by SMS. Vercel Analytics counts visits. That's the whole stack, and at this scale it's enough.

DELIBERATELY DEFERRED

Recorded in the plan, not forgotten: a member-level "forgot my PIN" (today recovery is per-family), head-of-household reassignment, a global family-creation cap, audio-only quick-join for bad connections, and per-family theming. Each is waiting on a real need, not a roadmap slide.

WHAT THIS PROJECT IS REALLY ABOUT

Callmata is small on purpose — roughly 8,000 lines of TypeScript, one Redis, one repo. The engineering interest is in the constraints: software for users who cannot troubleshoot, on a budget of approximately zero, with a public registration surface and no ops team. Every design above — renting the media stack, ringing over two channels, fail-loud guardrails, migrations absorbed at the read path — is the same answer to the same question: **what still works when nobody is watching it?**