Callmata

Architecture — In Plain English

The non-technical companion to the engineering design doc: what Callmata does, how it rings a phone without an app store, and the reasoning behind the design decisions — written in plain language, with the technical terms explained as they come up.

Brian Foster · **callmata.com**

August 2026 · v0.5.1 — A personal project, built for my own family. Not a commercial product.

SECTION 1

What it is

Video calling is a solved problem for most people, but not for a whole family at once. FaceTime only works if everyone is on Apple devices. Zoom means finding a meeting link. WhatsApp requires contacts to be set up correctly on both ends. Every mainstream option quietly assumes the person on the other end can troubleshoot when something goes wrong — and the family members I most wanted on these calls are exactly the ones who can't.

Callmata reduces the whole experience to three taps: open the icon on your home screen, tap your name, tap the person you want to call. Their phone rings twice over — a notification that behaves like an incoming call, and a plain text message saying who wants to chat. One tap on either, and they're in the family's private video room. There are no accounts, no passwords, no links to find, and nothing to download from an app store.

It began as a hard-coded app for my own family of five. It has since become a **multi-tenant platform** — one running copy of the software that serves many independent families. Any family can register with a phone number, receive its own join code and its own private video room, and manage its own members. Each family is completely invisible to every other family on the system.

3	1	2	~$0	0
TAPS FROM HOME SCREEN TO RINGING	DEPLOYMENT, MANY FAMILIES	RING CHANNELS — TEXT + PUSH	MONTHLY COST TO OPERATE	PASSWORDS, ACCOUNTS, OR INSTALLS

SECTION 2

What it's built with

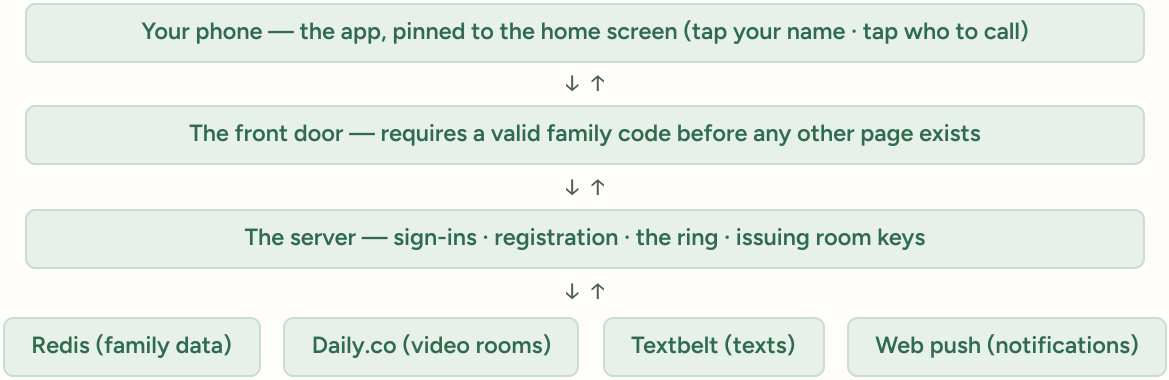
The stack is deliberately unexciting — the interesting decisions are about which problems I chose *not* to solve myself. Here is each piece and the role it plays.

PIECE	WHAT IT IS	WHAT IT DOES
The application	Next.js (TypeScript)	One codebase serves the public site, the app, and all of the server-side logic. Anything sensitive — PINs, API keys — runs on the server, where the browser can never see it.
The video	Daily.co	The video call itself is a rented service, embedded in my page. This is the most important decision in the project — Section 6 explains why.
The storage	Redis (a key-value database)	Every family's data — members, sessions, notification settings — lives in one small database, filed under that family's code so nothing can mix between households.
The texts	Textbelt	Sends the "so-and-so wants to chat" texts and the verification codes. Why this service and not Twilio comes down to a regulatory detail (Section 7).
The notifications	Web push	The pop-up-and-vibrate half of the ring. Free and built into every phone — but, as it turned out, not reliable enough to trust alone (Section 4).
Abuse protection	Cloudflare Turnstile + rate limits	Protects the public sign-up page, which by its nature sends a text message to any phone number entered into it (Section 8).
The hosting	Vercel	Where the site runs. Code deploys automatically when pushed; there is no server for me to maintain.

SECTION 3

The shape of it

The structure follows one rule: **your phone talks only to my server, and my server talks to everything else.** The browser never holds an API key, a password, or anyone's phone number. All of the sensitive material — the video service's key, the texting credentials, every member's PIN — stays server-side, and a phone receives only what it needs for the next few minutes.



Each family's records are filed under its own code – the code is not just how you get in, it's how everything is organized.

SECTION 4

What happens when you tap "Call"

The entire product comes down to the next fifteen seconds. Step by step:

1. **You're already signed in.** Signing in means tapping your name (shown as large initials, no typing) and entering a short PIN on an oversized on-screen number pad — the phone's keyboard, with its autocorrect and mode switches, never appears. A sign-in lasts thirty days, so most days no one sees this screen at all.
2. **You tap the person.** One request goes to the server: ring this member of my family.
3. **Their phone is signaled twice, in parallel.** A push notification styled like an incoming call — it stays on screen, vibrates in a ring pattern, and offers a "Join call" button — and a plain text message: "Callmata: Brian wants to chat."
4. **One tap lands them in the call.** No menus, no meeting ID. Behind the scenes, their phone requests a room key from the server — a temporary, named pass into the family's private video room — and the call opens directly.

WHY IT TEXTS AND PUSHES

The original plan was push notifications only — they're free and built into every phone. Testing on our family's actual Android devices changed that plan: **Android aggressively delays or drops notifications to save battery**, differently on every manufacturer's phones, and a website has no way to prevent it. A delayed "your order shipped" is harmless; a delayed "your daughter is calling" is the product failing at its one job. So the roles are inverted from what you might expect: **the text message is the reliable, primary ring, and the push notification is the free enhancement** that makes it feel like a real incoming call when it does arrive. Both are sent simultaneously — the server never waits on one channel before trying the other.

Two details in this flow are worth knowing about. First, when someone taps the notification, the app sends itself the "open the call" instruction through two different mechanisms, because iPhones running a pinned web app sometimes ignore one of them. Whichever instruction arrives first wins; joining a call twice is harmless by design, so the race resolves itself. Second, families share devices — there is usually a communal tablet somewhere — so a device only rings for the person who signed in on it most recently. Without that rule, a shared tablet would ring for every member of the family at once.

SECTION 5

The family code is everything

Callmata has no user accounts. The unit of identity is the **family**, and each family has a short join code — six characters, drawn from an alphabet with no 0-versus-O or 1-versus-l ambiguity, because these codes get read aloud over the phone. That one code serves three purposes at once: it's what you share with relatives so they can get in, it's the filing label for everything the family owns in the database, and it's stamped into every sign-in so the server always knows which family's data a request may touch.

Access is two doors deep. The first door is the code: until a valid family code has been entered, every page and every API endpoint refuses to respond. The second door is your name and PIN: the code gets you to your family's sign-in screen, and the PIN proves which member you are. The combination means that even someone who learns a family's code still can't impersonate a specific member.

A BUG THAT WAS WAITING FOR FAMILY #2

When the app served a single family, a sign-in didn't need to record *which* family it belonged to — there was only one possibility. The moment the app became multi-family, that shortcut turned into a genuine hole: a valid sign-in from family A, presented at family B's door, would have been accepted. The fix stamps the family code inside the sign-in token itself, and the server **rejects** a mismatch rather than correcting it to the current family. The lesson worth generalizing: converting single-tenant software to multi-tenant doesn't so much create new security requirements as **activate requirements that were always there, invisibly unmet**, because with one tenant there was no wrong answer to give.

How the original family moved in

My family predates the sign-up flow — our members were originally defined in the server's configuration, not the database. Rather than run a one-time migration script, I made the storage layer handle it: the first time anything asks for the original family, it is copied into the real database and served like any other tenant from then on. The

same pattern has absorbed every schema change since — when the "head of household" concept was added, older records simply gain the field the next time they're read. No data migration has ever been run against this app; records upgrade themselves the moment they're touched.

SECTION 6

The part I deliberately didn't build

The most consequential engineering decision in this project is about restraint. Real-time video is notoriously difficult to do well — not the demo, but the long tail: unusual home routers, weak cellular connections, dozens of Android variants, and whatever mobile Safari changes next. And Callmata's users cannot file a useful bug report; "the video is black again" is all the diagnostic information I would ever get.

So that entire problem is rented. Each family's calls run in a private room on Daily.co, a service whose whole business is making browser-based video work reliably across devices, using their pre-built call interface embedded in my page. What the app keeps for itself is **control of the door**:

- **Rooms are created by the server.** When a family completes registration, the server provisions their private room automatically. If the video service's API key is missing, registration fails with an error rather than creating a family whose room doesn't work.
- **Room keys are issued per join.** Entering a room requires a temporary, named token minted by the server; the API key that authorizes all of this never reaches a browser. Without a token, the private room cannot be entered.
- **The pre-join screen is removed.** Daily normally asks "Ready to join?" before entering a call — one extra question that adds nothing for this audience — so the server disables it for every room. You tap, and you're in the call.

> **A DEBUGGING STORY — THE DUPLICATE CALL WINDOW**
>
> During development I started seeing two call windows stacked on top of each other. The cause was an interaction between two facts: the embedded call is an iframe that must exist exactly once, and React (the UI framework) deliberately initializes every screen twice in development to expose exactly this kind of lifecycle bug. The fix makes every new call window wait for any previous one to be fully torn down before creating itself. A related lesson from device testing: iPhone Safari silently refuses to run WebRTC (the browser's video-calling machinery) over a connection that isn't fully secure, which produces a blank screen with no error. The app now detects that situation and explains it in a readable sentence instead.

SECTION 7

The texting layer

Sending a text message from software is technically trivial. What I didn't anticipate was the regulatory side: US carriers now require businesses to register before sending application-generated texts — a process called **10DLC**, involving forms, fees, and campaign approval. That framework makes sense for companies sending marketing texts at scale, and no sense for an app that texts one family. The major providers, Twilio and Telnyx, both require it.

Textbelt is a small paid service aimed at exactly this situation: personal-scale texting, no business registration, a few cents per message. It is the app's only texting service, and the sending code is a single small function — so if Textbelt ever disappears, substituting another provider is a contained, afternoon-sized change.

THE TEXTING LAYER, COMPLETE

```
# One provider, one function.
send a text(to, message):
    if Textbelt is configured:
        send it
    # in local development with nothing configured: print the code to the console
```

Textbelt comes with one real limitation worth being upfront about: **it blocks links in messages by default**, as an anti-spam measure. For most applications that would be a serious constraint — no "tap here to sign in," no links routing users back into the product. For Callmata it costs nothing, and that's a consequence of how the app is shaped rather than luck: the app is already pinned to the home screen and already signed in, so a call alert doesn't need to take anyone anywhere ("Callmata: Brian wants to chat" is the complete message), and verification codes are six digits you type, not links you follow. I'd argue the limitation is actually an asset. A plain text with no link is the opposite of what phishing looks like — and the last thing I want is to teach the most scam-targeted people in my life that tapping links in text messages is normal.

One more design choice in this layer: the verification codes the app texts you are **never stored anywhere**. The server hashes the code into a sealed, signed, ten-minute credential that is kept by your own browser, and checks what you type against that. There is no table of active codes to steal, because no such table exists.

SECTION 8

Keeping strangers out

Opening the app to public sign-up created a problem the private version never had: two pages on the open internet that will **send a text message to any phone number typed into them**. Unprotected, that is both a free text-spam service and a way to probe which phone numbers are registered. Four defenses are stacked on those pages:

1. **A bot check** (Cloudflare Turnstile, the kind that rarely shows a puzzle) — verified on the server, and configured to *fail loud*: if the bot check is somehow unconfigured in production, those pages refuse to operate entirely. A configuration mistake can take sign-up down, but it can never quietly leave it unprotected.
2. **A rate limit.** Five attempts per hour per network address. The counter is written so that even with the server running as many short-lived parallel instances (which is how Vercel works), two instances can't accidentally grant the same caller a fresh allowance.
3. **No fishing for numbers.** The "forgot your code" page returns an identical response whether or not the phone number belongs to a real family — only a genuine match actually receives a text. An outside observer learns nothing either way.
4. **An alert to the operator.** If someone trips a rate limit, I receive a text about it — capped at one per hour so that an attack can't flood my phone either. This app has no monitoring dashboard or on-call rotation; at this scale, a text to the person who runs it is the appropriate tool.

Sign-in itself is throttled as well, an addition made after re-reviewing this design: wrong PIN guesses lock the endpoint after ten failures in fifteen minutes. Only failures count — a family mistyping PINs on shared Wi-Fi will never encounter the limit — and tripping it sends me the same alert. The reasoning: a 4-digit PIN has only ten

thousand possible values, so an endpoint that allowed unlimited guessing was a genuine gap. The texted verification codes are covered the same way, with a cap of ten guesses per ten minutes on every endpoint that accepts one.

Within a family, permissions are modeled on how families actually operate rather than on enterprise roles. Any signed-in member can add a member or change their own PIN. Resetting *someone else's* PIN is reserved for the **head of household** — the person whose phone number registered the family — because one person set the system up, and that's who everyone turns to when something needs fixing.

One gap is left open deliberately, and I'd rather name it than pretend otherwise: the rate limits are per-address, and there is no global cap on total sign-ups. For an unadvertised app this is an acceptable risk, and it sits at the top of the list to revisit if that ever changes. A known, documented gap is a far better position than an unknown one.

SECTION 9

Designing for grandparents (the actual point)

Everything above is plumbing. This is the product. The design bar for Callmata is not "easy to use" — it is **"cannot be gotten wrong by someone who is done learning new software."** Every feature had to clear that bar:

NO TYPING, ANYWHERE

Names are large tappable circles. The PIN pad is custom-built and oversized, and never invokes the phone's keyboard with its autocorrect and hidden modes. The app greets the last person who used it by name. When the camera fails, the error message is a sentence a person would say — "another app is using your camera" — rather than whatever the browser reported.

SET UP ONCE, BY WHOEVER VISITS

Callmata installs like an app but is actually a pinned website (a PWA, in web terms). The setup screen detects iPhone versus Android — including iPads that report themselves as Macs — and shows the correct "Add to Home Screen" steps for that device. The assumption is that a tech-comfortable relative performs this step once, in person; from then on it's simply an icon.

LOSING THINGS IS THE NORMAL CASE

No one remembers a code they entered once, years ago, on a phone someone else set up. Recovery therefore works from the one identifier families reliably keep — the phone number. Prove you own it via a texted code, and the family code comes back with a PIN reset included. The forgotten-credential path is a designed feature, not a support request.

SMALL COURTESIES

"Not your family?" backs out of a mistyped code. "Open the room without ringing" lets someone wait in the call without alerting anyone. The home screen shows who is already in the call and whose alerts are on, so tapping "Call" is never a guess. Even the visual design — warm paper tones, deep green, a serif typeface — is chosen to read calm rather than technical.

SECTION 10

What it costs, and what's next

THE OPERATING COST

Nearly nothing, by design. The hosting, database, and video service all run on free tiers at family scale, and texts cost a few cents each. That matters more than it might seem: this software needs to keep working for years without a monthly bill prompting the question of whether it's worth renewing.

RUNNING IT

A live status panel (secret-gated in production) shows who is in the call and who is waiting — that is the entire operations dashboard. Abuse alerts arrive by text. Code deploys itself when pushed. The full extent of my operational duties is reading the occasional alert and joining the calls.

The deferred list, written down so it stays honest: a member-level "forgot my PIN" flow (today recovery works at the family level), a way to transfer head-of-household if the original registrant leaves, a global cap on sign-ups, an audio-only mode for poor connections, and per-family themes. Each item is waiting on a demonstrated need. A large part of the discipline in a personal project is the features you decline to build.

WHAT THIS PROJECT IS REALLY ABOUT

Callmata is small on purpose — roughly eight thousand lines of TypeScript, one database, one repository. The interesting constraints were never technical; they were human: users who cannot troubleshoot, a budget of approximately zero, and no one watching the system but me. Every significant decision described above — renting the video stack, trusting texts over push notifications, making misconfiguration fail loudly, letting data upgrade itself on read — is the same answer to the same question: **what still works when nobody is watching it?** And the measure I care about most: my family actually uses it, every week.