Full-Stack Web Application in One Weekend — For $7.68

Pallas Advisory | Community Web App Case Study

The Situation

A community organization in the Pacific Northwest had a simple problem: dozens of local groups were hosting events, and there was no central place to find them.

I took this on as a pro-bono project — a community contribution, not a client engagement. But the methodology and the result demonstrate something that applies directly to paying work: what's now possible when you combine AI-assisted development with modern free-tier infrastructure.

Every group maintained its own website, its own Facebook page, its own email list. If you wanted to know what was happening in the community this week, you had to check twelve different places. Events fell through the cracks. Organizations doing similar work didn't know about each other's activities. Community members who would have shown up never heard about it.

The existing attempts at a shared calendar were incomplete and didn't allow community members to submit events. The result was a classic coordination failure — not because people didn't care, but because nobody had built the infrastructure for it.

There was a catch: the person who would administer this system was a volunteer with limited technical experience. Any solution that required logging into a dashboard, navigating a CMS, or learning new software was dead on arrival. The workflow had to be as simple as checking email.

There was another catch: the local political climate meant that the administrator's personal identity needed to be completely shielded from the public-facing system. This wasn't a nice-to-have. It was a safety requirement.

Why Not Use an Existing Platform?

This was a volunteer organization with no budget. Every option was evaluated against three hard constraints: zero ongoing cost, zero technical skill required for administration, and complete separation between the admin's identity and the public-facing system.

Platform	Why It Didn't Fit
WordPress / Squarespace	$4–16/month recurring cost kills volunteer projects. Still requires dashboard management — training burden the admin didn't need.
Google Calendar (embedded)	No community submissions, no design control, and someone's personal account is still attached.
Facebook Events / Eventbrite	Data lives on someone else's platform. No single neutral URL to hand the community.
Freelancer build	$2,000–5,000 and weeks of timeline. Not realistic for a volunteer org.

What they actually needed was a fully custom web application — purpose-built for this community's specific needs — that costs nothing to operate and requires the administrator to do nothing more complex than tap a button in an email.

What I Built

A full-stack web application with three components: a public-facing event calendar, a community submission form, and a one-click email approval workflow.

The public calendar displays upcoming events organized by date, filterable by organization. It's server-rendered for performance and SEO — when someone shares the link on Facebook or Nextdoor, the preview card shows the site name, description, and a proper image. Events stay visible through the end of the day they occur (in the local timezone, not UTC — a detail that matters more than you'd think).

The submission form lets any community member submit an event. Server-side rate limiting prevents spam (three submissions per hour per IP address), and a honeypot field catches bots. No account required. No CAPTCHA. Just fill out the form and submit.

The approval workflow is where the real design thinking lives. When someone submits an event, the admin receives a formatted email with all the event details and two links: Approve or Reject. One tap. That's it. No login, no dashboard, no app to install. Behind the scenes, each link contains a cryptographic token (SHA-256) that

expires after use — so approval links can't be reused or guessed.

The admin's identity is completely separated from the system. Approval emails route through an organizational email address with forwarding configured at the domain level. The admin's personal email appears nowhere in the codebase, the documentation, or any public-facing element. This was an architectural decision made on day one, not a patch applied later.

The visual design reflects the region — earthy tones, serif headings, imagery inspired by the local landscape. It looks like it belongs to the community, not like a generic template.

The AI-Assisted Development Workflow

This project was built using a methodology I'm now applying to client work: structured specification documents executed by AI coding tools, with human review at every stage.

The workflow: I made all the strategic and architectural decisions — tech stack, database schema, security model, visual design direction, admin workflow design. Then I wrote precise specification documents covering each concern: product requirements, visual design brief, security hardening checklist, domain and deployment configuration. Each document included explicit "what NOT to do" sections and ordered dependencies.

The AI coding tool built to spec. I reviewed every output for security and correctness.

This is important to understand because it's where the speed came from. I didn't type every line of code — I designed the system, specified it precisely, and used AI tools to execute the implementation. That's a fundamentally different capability than either "I code everything by hand" or "I let AI do whatever it wants." The spec quality directly determines the output quality. Vague specs produce vague code. Precise specs produce precise code.

The reason this matters for clients: this workflow means custom software can now be built at a speed and cost that wasn't possible two years ago. The bottleneck has shifted from implementation to design — and design is where I spend my time.

What Got Built in a Weekend

The system launched with 13 local organizations already seeded in the database and 16 real events scraped from their websites — so the calendar wasn't empty on day one. A branded QR code was produced for physical flyers. An admin onboarding guide was written specifically for the admin's needs — one printed page, clear visual layout, no jargon.

The technology:

Component	Technology	Cost
Frontend	Next.js (TypeScript), Tailwind CSS	$0/month (Vercel free tier)
Database	PostgreSQL (Supabase)	$0/month (free tier)
Automation	n8n workflow for email approval	$0/month (self-hosted)
Security	SHA-256 tokens, rate limiting, CSP headers, HSTS	Built-in
Domain	Custom .org domain	$7.68/year
Analytics	Privacy-friendly, no cookies	$0/month (Vercel Analytics)
Total		**$7.68 total. $0/month ongoing.**

Twelve database migrations. One serverless function. One automation workflow. Complete technical documentation. A product specification, a design brief, and a security review — all delivered alongside the working application.

The Numbers

Metric	Result
Time from first commit to production	One weekend
Total investment	$7.68 (domain registration)
Monthly operating cost	$0
Organizations in system at launch	13
Events seeded at launch	16
Admin workflow complexity	Tap one link in an email
Admin training required	One page, under 5 minutes
Ongoing technical maintenance required	None (build-and-handoff model)

What Was Hard

AI-generated code had security problems on the first pass

The AI coding tool produced functional code quickly — but the initial output had overly permissive database access policies. In a traditional development context, this is the equivalent of leaving the front door unlocked because it's easier than setting up keys.

Specifically: the first version created public read/write access on the submissions table through the database's row-level security policies. This meant anyone with the database URL could theoretically read or modify submissions directly, bypassing the application entirely. The fix was straightforward — remove all public-facing database access and route every write operation through the server — but the fact that it needed fixing at all is the point.

AI coding tools are fast. They are not careful. Security review of every piece of AI-generated code is non-negotiable, and this is a step I now build explicitly into every project workflow. The tool also marked database functions with incorrect volatility settings — functions that check the current time were labeled as returning constant values, which would have caused events to display on the wrong days.

These aren't exotic bugs. They're exactly the kind of mistakes that ship to production when AI-generated code doesn't get a human security review.

Designing for a non-technical admin was harder than the code

The technical build was the easy part. The hard part was designing a workflow simple enough that a busy volunteer — who didn't ask for this responsibility and has competing priorities — would actually use it.

Every feature decision was filtered through one question: does this require the admin to learn something new? If yes, cut it. The email approval workflow exists specifically because email is the one interface everyone already knows. No logins, no dashboards, no "admin panels."

The onboarding document went through multiple revisions to get it down to a single printable page. Not because the system is complex — it isn't — but because the moment a volunteer sees a multi-page instruction manual, they put it aside and never come back.

Planning for physical safety from the start

Shielding the administrator's identity wasn't a feature request — it was a constraint I identified during discovery. In politically active communities, the people doing coordination work can become targets for harassment. The architecture was designed from day one so that the admin's personal information exists nowhere in the public-facing system.

This is much easier to build in from the start than to retrofit. Once personal details appear in a codebase, a domain registration, or a public page, removing them completely is painful. The lesson: if there's any chance that identity protection matters, architect for it upfront.

Why This Matters Beyond One Community

Most small organizations — nonprofits, community groups, local businesses — hit the same wall. They need custom software, but custom software costs thousands of dollars and takes weeks. So they cobble together free tools that don't quite work, or they pay for SaaS subscriptions that cost more than they should for what they actually need.

What's changed is the economics. Modern infrastructure (managed databases, serverless hosting, free-tier cloud platforms) combined with AI-assisted development means that purpose-built software can now be delivered faster and cheaper than configuring an off-the-shelf tool to do something it wasn't designed for.

This project — a full-stack web application, custom-designed for a specific community's needs, with proper security, documentation, and a zero-cost operating model — was built in a weekend for under eight dollars. That's not a proof of concept.

It's in production.

The same approach applies to any organization that needs software tailored to their specific workflow — and for paying clients, the economics are even more compelling. The speed that delivered a free community tool in a weekend translates directly to faster timelines and lower costs on commercial projects. The methodology is the same; the scope and investment scale to the problem.

The question isn't whether custom makes sense anymore. The question is whether the person building it understands both the technology and the actual human problem it needs to solve.

The deliverable is a working system, not a strategy deck.

Does your organization have a workflow problem that off-the-shelf tools haven't solved? Let's talk about what a purpose-built solution would actually cost.
info@pallasadvisory.com