NEWS · OCTOBER 20, 2026

No es vibe coding: 58.588 líneas, 313 pruebas y 48 versiones en 12 días

Doce días, 58.588 líneas, 313 pruebas automatizadas, 34 comprobaciones de aislamiento, 52 pull requests y 48 versiones -- las cifras y las salvaguardas detrás de Hearth, y por qué el desarrollo rápido asistido por IA y la ingeniería cuidadosa no son opuestos.

By Dr. Holger Flick

Cada vez que enseño lo que construyo con IA, alguien hace la pregunta con educación, y otro la hace con algo menos de educación: «¿Eso no es simplemente vibe coding?». Es una pregunta justa. El término viene de Andrej Karpathy, que en febrero de 2025 describió una forma de programar en la que te «entregas por completo a las vibras, abrazas lo exponencial y olvidas que el código siquiera existe» -- aceptando cada cambio sin leer los diffs (Wikipedia recoge la publicación original y el debate posterior). Para un prototipo de fin de semana, es una forma perfectamente razonable de divertirse.

Para un software que guarda las pólizas de seguro, las facturas médicas y las contraseñas de una familia, no lo es. En mi artículo anterior presenté Hearth, el sistema doméstico que construí para mi esposa en doce días.

Este artículo es la otra mitad de la historia: las cifras y las salvaguardas que las hicieron posibles. Mi tesis es sencilla, y la respaldo con el propio repositorio. Rápido y cuidadoso no son opuestos -- la IA también abarató las partes cuidadosas.

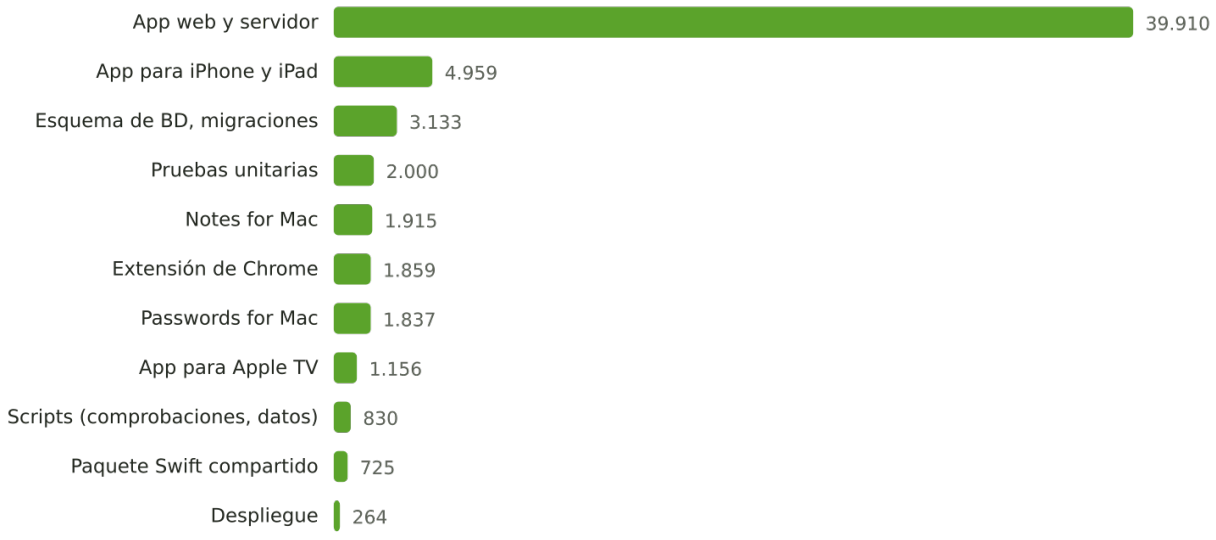
Al terminar sabrás exactamente qué se midió, cómo se organizaron las pruebas y las revisiones, cómo se protegen los datos y qué hice yo frente a lo que hizo la IA.

Las cifras

Todo lo que aparece en esta tabla sale directamente del repositorio Git, del ejecutor de pruebas y de los registros de mis sesiones de Claude Code. El recuento de líneas excluye las líneas vacías y el código generado.

Qué	Cantidad
Días de calendario, del primer prompt a la última versión	12 (del 7 al 18 de octubre)
Tiempo de sesión activa con la IA	unas 20 horas
Mis mensajes a la IA	unos 110
Líneas de código no vacías (TypeScript, Swift, SQL, JavaScript)	58.588
Tablas / migraciones de base de datos	44 / 30
Pull requests fusionados / versiones etiquetadas	52 / 48
Pruebas automatizadas (Vitest, extensión de Chrome, paquete Swift)	294 + 15 + 4 = 313
Comprobaciones del script de aislamiento entre familias	34
Triggers de base de datos que imponen quién es dueño de qué	49
Aplicaciones sobre un mismo backend	7

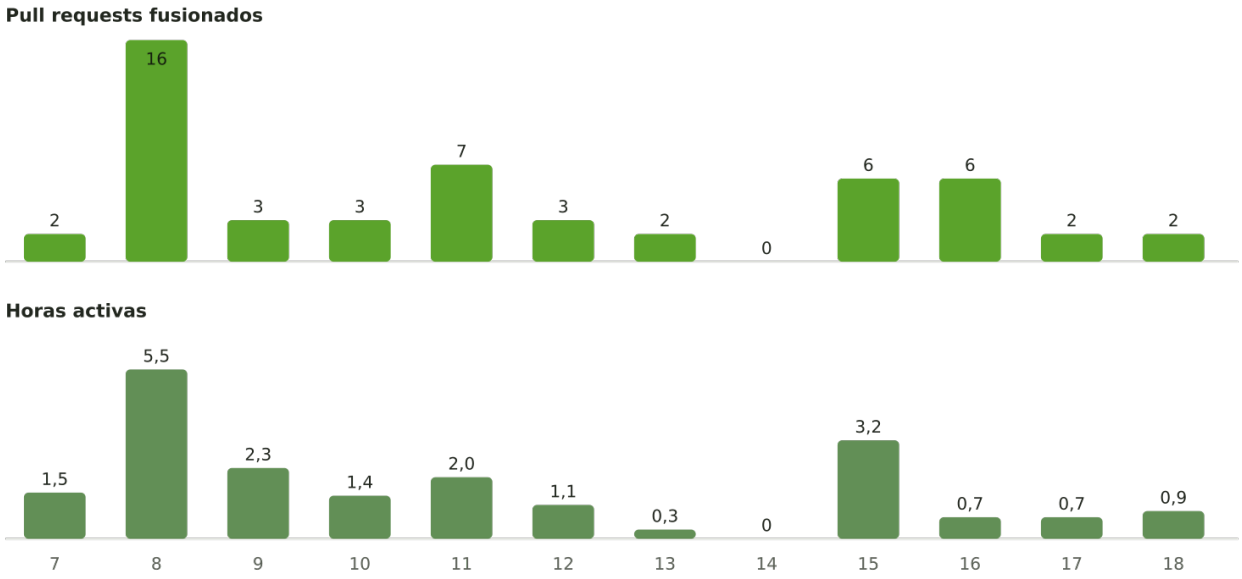
Así se reparten esas líneas entre las siete aplicaciones y sus partes compartidas. La aplicación web y su servidor llevan la mayor parte del peso, como debe ser: ahí viven todas las reglas, y todas las demás apps hablan con ella.



Líneas de código no vacías por parte de Hearth (58.588 en total)

La imagen día a día es más interesante que los totales, porque muestra el ritmo. Los pull requests llegaron en ráfagas, y las horas activas se mantuvieron modestas. El día de más actividad, el 8 de octubre, se fusionaron dieciséis pull requests en unas cinco horas y media de sesión -- desde el aislamiento entre familias y la Claude

API hasta una lista de documentos más tranquila --, después de haber usado la primera versión durante un día y saber qué quería cambiar.



Pull requests fusionados (arriba) y horas de sesión activa (abajo) por día, del 7 al 18 de octubre

Las «horas activas» cuentan el tiempo en que una sesión estaba ocupada, sin ninguna pausa de más de veinte minutos. A ratos corrieron varias sesiones en paralelo, y esos solapamientos se cuentan una sola vez.

> **Lo que las cifras no dicen**
>
> Veinte horas de sesión no son veinte horas de mi vida. Pensar, probar en el iPad y en el Apple TV, y convivir un día con la app antes de decidir qué cambiar ocurrió todo alrededor de ellas. Las líneas de código tampoco miden el valor; Dijkstra sostenía que deberían contabilizarse como «líneas gastadas».
>
> Aquí las uso por una única razón: dan una idea de la escala con la que pueden compararse los modelos de estimación clásicos, cosa que hice en el artículo anterior.

Salvaguarda 1: cada cambio es un pull request

El vibe coding no tiene historial; aceptas, ejecutas y sigues adelante. Hearth tiene 52 pull requests fusionados, y cada uno lleva una descripción escrita de qué cambió y por qué. Los más grandes también explican cómo se verificó el cambio -- la descripción de la revisión de seguridad, por ejemplo, termina con «tsc, biome, `pnpm build` y `pnpm check:isolation` (26/26) pasan», además de una copia de seguridad de la base de datos restaurada en una base de datos temporal y comparada con la de producción.

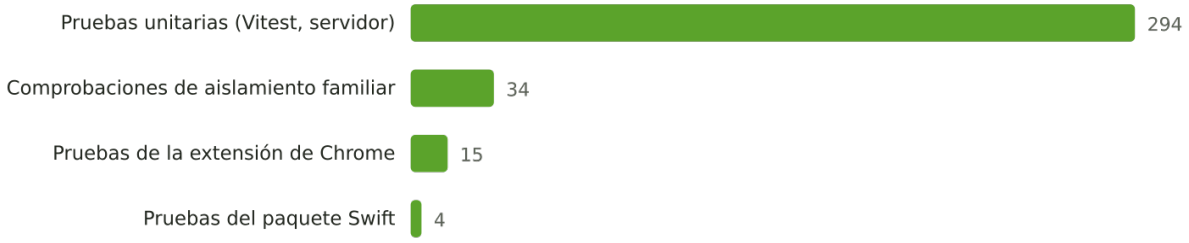
Los pull requests pequeños no son burocracia. Son lo que te permite encontrar el cambio que rompió algo, leer un diff en cinco minutos en lugar de una hora y revertir exactamente una cosa. El programa de investigación DORA lleva años diciendo que los lotes pequeños y las pruebas exhaustivas importan, y su informe de 2024 encontró que la adopción de IA iba de la mano de una *menor* estabilidad en las entregas. En mi opinión, ese hallazgo no es un argumento contra la IA. Es un argumento para mantener los lotes pequeños justo cuando la IA tienta a hacerlos enormes.

Salvaguarda 2: pruebas que encontraron errores reales el primer día

El 12 de octubre, el proyecto recibió sus pruebas unitarias: 186 al principio, 294 hoy, escritas con Vitest para las funciones puras de `src/lib` -- cálculo de fechas, calendarios recurrentes, reglas de negocio, parsers y guardas de solicitudes. Se ejecutan en medio segundo, así que nadie tiene motivos para saltárselas. Un detalle que me gusta bastante: la máquina de pruebas finge estar en Honolulu, que no es ni UTC (el servidor) ni Nueva York (el hogar). El código que olvida indicar una zona horaria falla ahí y no en producción.

La primera ejecución encontró cuatro errores reales, y cada uno se corrigió con una prueba que primero fallaba.

1. **Presupuestos en el día equivocado.** Dos presupuestos enviados a las 3 de la tarde y a las 9:30 de la noche, hora de Nueva York, cayeron en «rondas» distintas, porque a las 9:30 de la noche ya es el día siguiente en UTC. El presupuesto más barato, el de la tarde, se contaba como reemplazado.
2. **Una redirección abierta.** La función que protege el «volver a esta página después de iniciar sesión» aceptaba `/\t/evil.example`. Los navegadores eliminan los tabuladores y los saltos de línea, así que esa dirección llevaba a otro sitio.
3. **Nombres heredados.** Un parser aceptaba nombres que todo objeto de JavaScript hereda, como `toString`.
4. **Días que no existen.** La validación de fechas aceptaba `2026-02-30`.



Comprobaciones automatizadas en Hearth: pruebas unitarias, comprobaciones de aislamiento, pruebas de la extensión y de Swift

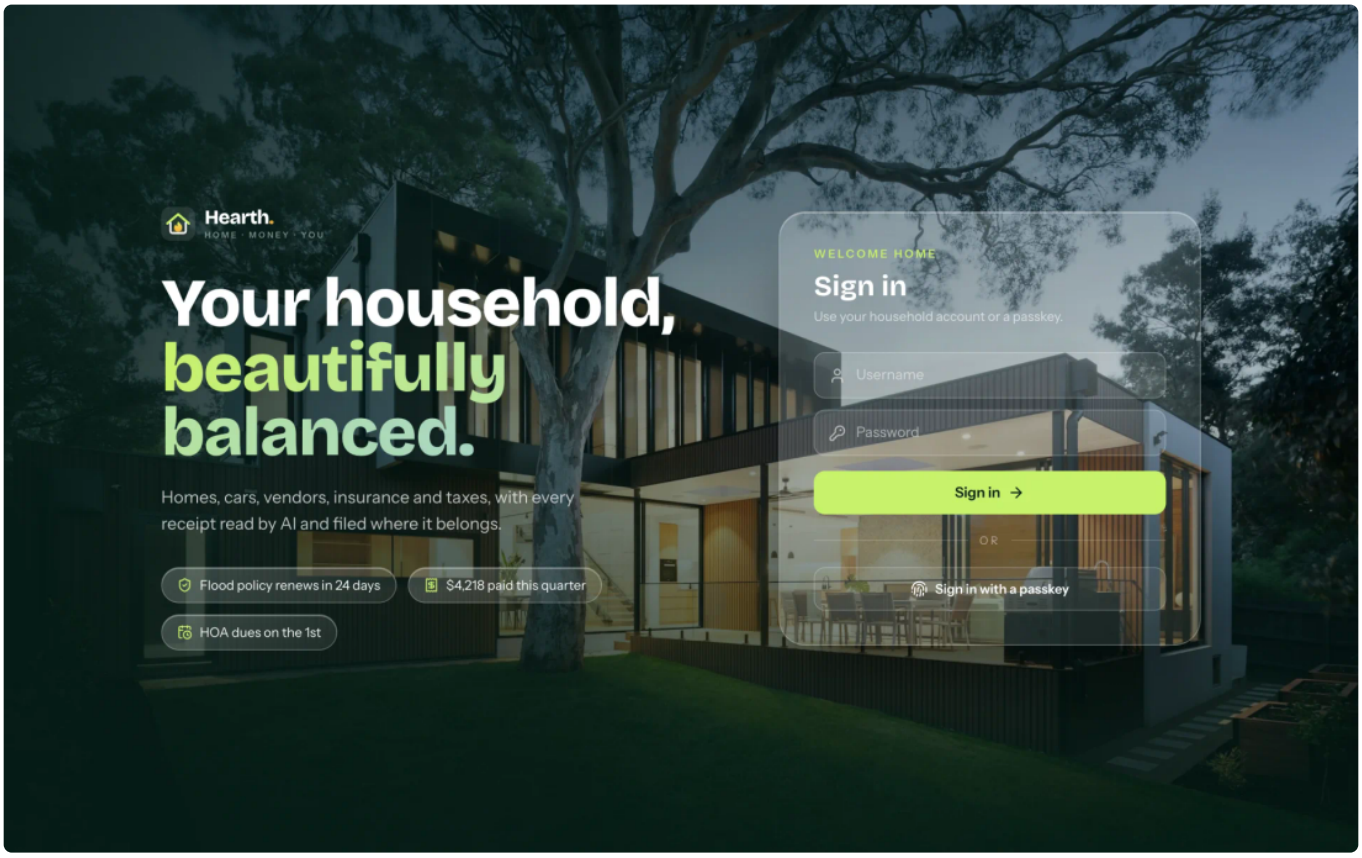
Ese mismo día, las pruebas se convirtieron en ley. El script de despliegue las ejecuta antes de compilar nada y se niega a publicar si alguna falla, despliegue quien despliegue. Para demostrar que la barrera funciona de verdad, rompimos a propósito la función `relativeDays` y vimos cómo el despliegue se detenía con «Unit tests failed: not deploying». Luego la arreglamos de nuevo. Las instrucciones de publicación también prohíben saltarse, borrar o relajar una prueba para sacar una versión adelante -- una regla escrita tanto para la IA como para mí.

> **Dónde se detienen las pruebas**
> Las pruebas unitarias cubren la lógica, no las pantallas. Todavía no hay pruebas end-to-end en el navegador, y las apps en Swift solo tienen un puñado de pruebas en su paquete compartido. Cada cambio de pantalla se comprobó ejecutando la app -- en el navegador, en los simuladores, en el iPad real -- y los pull requests lo dicen, pero eso es verificación manual, no automatización. Para un producto con muchos clientes en lugar de un solo hogar, las pruebas end-to-end serían lo siguiente que añadiría.

Salvaguarda 3: un inicio de sesión bien hecho

El inicio de sesión es un buen lugar para distinguir el trabajo profesional de una demo, porque una demo se queda en «me deja entrar». Hearth ofrece contraseñas y passkeys, el estándar de inicio de sesión resistente al phishing de la FIDO Alliance, basado en WebAuthn del W3C (implementado con SimpleWebAuthn). Las contraseñas se guardan con bcrypt; un nombre de usuario desconocido se comprueba contra un hash ficticio

del mismo coste, así que la respuesta tarda exactamente lo mismo que con uno real y no revela qué cuentas existen.



La página de inicio de sesión: contraseña o passkey, con limitación de intentos detrás de ambas (instancia de demostración).

En los detalles está el trabajo. Los intentos fallidos se limitan por nombre de usuario y por dirección, y los contadores viven en la base de datos, de modo que un reinicio no los pone a cero. Los tokens de sesión son aleatorios y solo se guardan como HMAC, así que una copia de la base de datos no permite iniciar sesión a nadie, y la cookie es `httpOnly`, `secure` y `sameSite`. Cada manejador de rutas comprueba el origen de la solicitud. Cambiar la contraseña cierra tus otras sesiones y todas las apps conectadas, añadir una passkey exige primero la contraseña, y cuando un administrador restablece una contraseña también se eliminan las passkeys. Y cada server action y cada ruta de la API comprueban al usuario por sí mismas, en lugar de confiar en que algún middleware ya lo hizo.

Salvaguarda 4: una familia nunca puede ver a otra

Hearth es multi-tenant: varias familias y, dentro de cada familia, registros compartidos y registros privados de un solo miembro. Un error aquí no es un bug; es un incidente de privacidad. Por eso el aislamiento se impone tres veces, en tres niveles distintos.

Tres capas independientes mantienen los registros de una familia lejos de otra

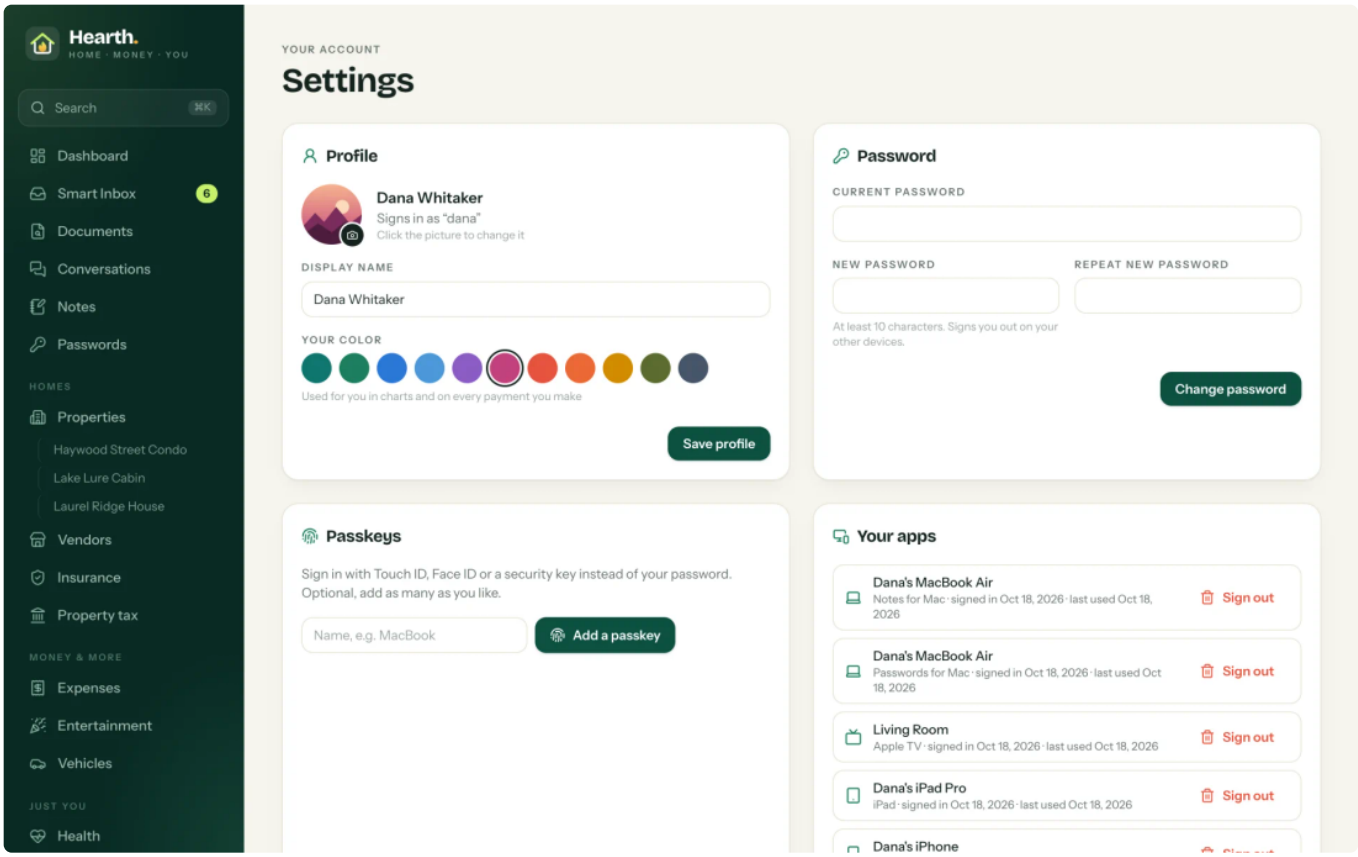
La capa interior es un cliente de base de datos que añade automáticamente las reglas de familia y de miembro a cada consulta, incluidas las relaciones anidadas y los recuentos, para que un desarrollador -- o una IA -- no pueda olvidarlas. La capa intermedia no confía en la interior: cuarenta y nueve triggers en PostgreSQL verifican la propiedad al confirmar cada transacción. La capa exterior no confía en ninguna de las dos: un script crea una familia desechable, prueba 34 maneras de leer, cambiar o vincular registros al otro lado de la frontera y después la elimina de nuevo.

En mi opinión, esta es la diferencia más importante entre un prototipo y un producto. Cualquiera puede escribir `where familyId = ?`. La pregunta es qué pasa el día en que alguien se olvida.

Salvaguarda 5: los secretos siguen siendo secretos

La bóveda de contraseñas cifra cada campo de una entrada -- título, dirección, nombre de usuario, contraseña y notas -- con AES-256-GCM, con una clave derivada de un secreto del servidor. Cada valor queda vinculado a su familia, su entrada y su campo como datos autenticados adicionales. Si copias una contraseña cifrada de una fila a otra, ya no se abre. Las contraseñas anteriores quedan en un historial, cifradas de la misma manera, así que una contraseña antigua nunca se pierde.

Las apps siguen la misma idea de mínimo privilegio. Cada una inicia sesión por separado con un código que se confirma en la web, y su token solo se guarda como HMAC. Cada tipo de app solo puede hacer aquello para lo que fue creada: la app para Apple TV y la de iPhone y iPad solo leen y nunca cambian nada, Notes for Mac solo ve notas y la extensión de Chrome solo ve contraseñas. Dentro de la extensión, solo el proceso en segundo plano guarda el token, y solo le responde a una página web sobre el propio sitio de esa página, tras un clic real.



Ajustes ' Tus apps: cada app con sesión iniciada, con su tipo, cuándo inició sesión y cuándo se usó por última vez -- y cada una puede cerrar sesión por separado.

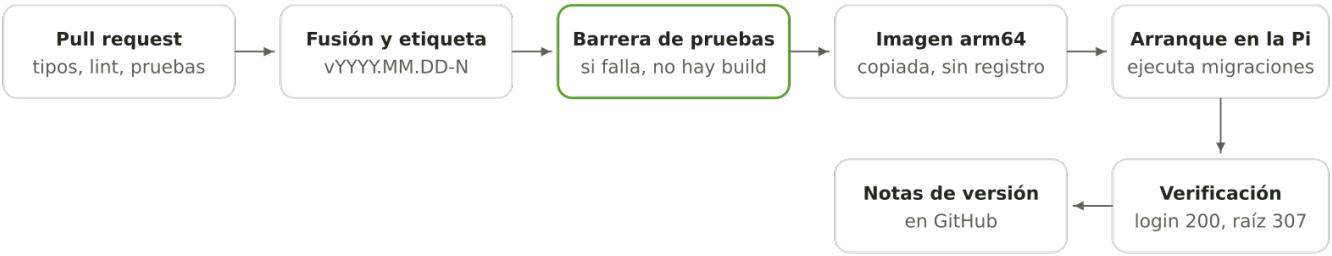
Salvaguarda 6: una revisión que encontró lo que la IA hizo mal

El 11 de octubre le pedí a la IA que revisara su propio código en busca de problemas de autorización, de privacidad entre miembros y de seguridad -- no como una formalidad, sino con el encargo explícito de encontrar problemas. Encontró problemas reales, y para eso se hace. Entre ellos había un agujero de **cross-site scripting almacenado**: una subida de archivo podía elegir su propio tipo de contenido, y un archivo manipulado podría haber ejecutado código en la app. También encontró sitios donde editar un registro privado podía hacerlo visible para el resto de la familia, y un informe de uso de IA que mostraba al administrador los títulos de los documentos.

Todo eso se corrigió en el mismo pull request, junto con la limitación de intentos en la base de datos, el cierre de sesiones al cambiar la contraseña y las copias de seguridad nocturnas. Quiero ser honesto sobre lo que esto significa. La IA escribió esos errores, y la IA los encontró -- porque yo pedí una revisión y la traté como una barrera, no como un ritual. No es distinto de cómo trabaja un buen equipo con desarrolladores junior. Nadie publica nada sin un segundo par de ojos.

Salvaguarda 7: versiones y operación

Hearth corre en una Raspberry Pi 5 en casa, en un contenedor Docker detrás de Caddy, que se ocupa de los certificados HTTPS. Cada versión sigue el mismo pipeline, y ese pipeline está escrito como una skill que la IA ejecuta paso a paso, de modo que una versión publicada una noche de cansancio es idéntica a una publicada una mañana descansada.



Del pull request a producción: cada versión sigue el mismo camino

Las últimas cinco imágenes publicadas se quedan en la Pi, así que una vuelta atrás es una línea en el archivo de compose. La base de datos se vuelca cada noche en un bucket de almacenamiento aparte, con copias diarias y mensuales, y antes de fiarme de ella restauré una copia en una base de datos temporal y la comparé con la de producción. Esa lección la aprendí por las malas, como conté en My Server Was Compromised: la arquitectura decide el radio de la explosión mucho antes de que algo salga mal.

Qué hizo la IA y qué hice yo

El reparto del trabajo es la parte que a la gente más le cuesta creer, así que seré concreto. En doce días escribí unos 110 mensajes; la IA respondió con unas 3.900 llamadas a herramientas -- leer archivos, escribir código, ejecutar pruebas, compilar, desplegar. Escribió casi todo el código, las pruebas y las descripciones de los pull requests.

Lo que no hizo fue decidir. Yo elegí la arquitectura y el stack: Next.js 16 con React 19, Prisma 7 sobre PostgreSQL, TypeScript 7 con su nuevo compilador nativo, SwiftUI para las apps de Apple y una extensión de Chrome con Manifest V3. Yo decidí que el aislamiento se impone en la base de datos y no solo en el código, que las contraseñas quedan vinculadas a su fila, que cada app recibe solo su propia parte y que las pruebas son la barrera de cada versión. Probé en dispositivos reales, dejé por escrito las reglas que la app debía seguir («primero las fechas, lo más reciente arriba», «nada de desplegables simples, siempre con búsqueda») y dije que no -- a menudo. Cuando el inicio de sesión del iPad se parecía a la pantalla del código del Apple TV, fue una persona quien se dio cuenta.

> **La investigación dice: depende del desarrollador**
> No quiero fingir que la evidencia apunta en una sola dirección. En un ensayo controlado aleatorizado de METR, dieciséis desarrolladores experimentados de código abierto fueron un 19 % más lentos con las herramientas de IA de principios de 2025 en sus propias bases de código grandes, mientras creían que iban más rápido. La propia METR enumera los límites -- una muestra pequeña, un solo tipo de trabajo, herramientas que han cambiado desde entonces -- y ha publicado datos más recientes. La lección que saco no es que la IA sea lenta o rápida. Es que la ganancia depende de quien la dirige: de con cuánta precisión especifica, con cuánto rigor revisa y de si trata las pruebas y las revisiones como barreras. Sin eso, obtienes vibras. Con eso, obtienes Hearth.

Lo que esto demuestra

Rápido y cuidadoso ya no son opuestos. La misma IA que escribió 58.588 líneas en doce días escribió también las 313 pruebas, las 34 comprobaciones de aislamiento, los 49 triggers y la revisión de seguridad que mantuvieron honestas esas líneas -- porque se le pidió, y porque nada llegó a producción sin superarlas.

La velocidad vino de las herramientas. El cuidado vino de treinta años sabiendo qué falla en producción, y de negarme a saltarme las partes aburridas.

> La IA no eliminó la ingeniería. Eliminó la excusa para saltársela.

Si quieres esta velocidad para tu propio producto -- una aplicación Delphi que necesita un futuro moderno, o un sistema que se adapte a tu negocio y no al revés --, echa un vistazo a lo que construí, a lo que ofrezco y luego hablemos.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —