NEWS · OCTOBER 20, 2026

# Não é vibe coding: 58.588 linhas, 313 testes e 48 versões em 12 dias

Doze dias, 58.588 linhas, 313 testes automatizados, 34 verificações de isolamento, 52 pull requests e 48 versões -- os números e as salvaguardas por trás do Hearth, e por que desenvolvimento rápido assistido por IA e engenharia cuidadosa não são opostos.

By Dr. Holger Flick

Sempre que mostro o que construo com IA, alguém faz a pergunta com educação, e outra pessoa a faz com menos educação: "Isso não é só vibe coding?" É uma pergunta justa. O termo vem de Andrej Karpathy, que descreveu em fevereiro de 2025 uma forma de programar em que você "se entrega totalmente às vibes, abraça as exponenciais e esquece que o código sequer existe" -- aceitando cada mudança sem ler os diffs (a Wikipedia reúne o post original e o debate desde então). Para um protótipo de fim de semana, é uma forma perfeitamente razoável de se divertir.

Para um software que guarda as apólices de seguro, as contas médicas e as senhas de uma família, não é. No meu post anterior, mostrei o Hearth, o sistema doméstico que construí para a minha esposa em doze dias.

Este post é a outra metade dessa história: os números e as salvaguardas que os produziram. A minha tese é simples, e vou sustentá-la com o próprio repositório. Rápido e cuidadoso não são opostos -- a IA também barateou as partes cuidadosas.

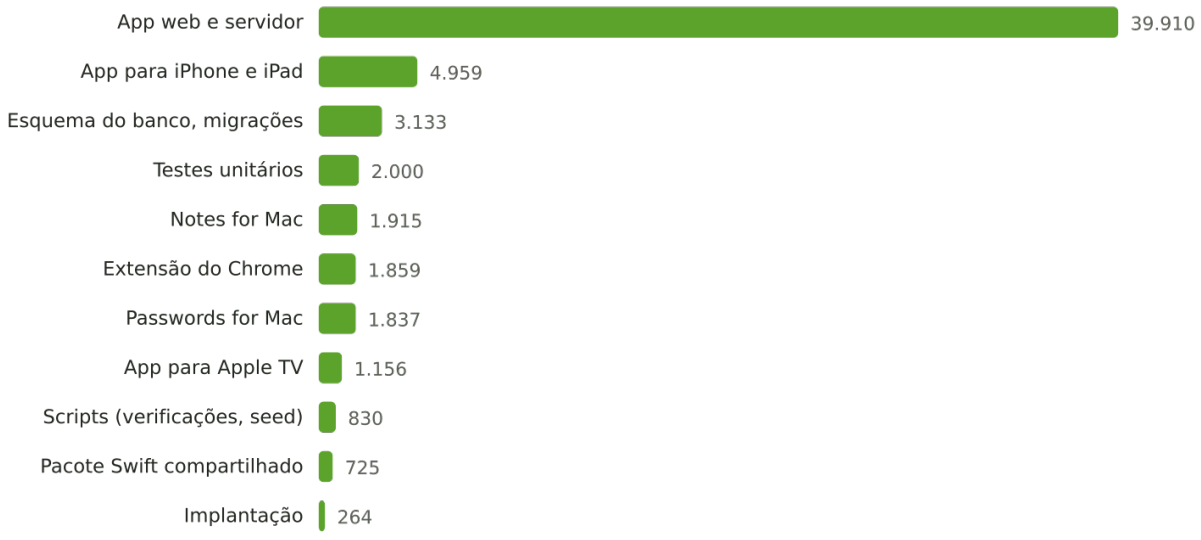
Ao final, você vai saber exatamente o que foi medido, como os testes e as revisões foram organizados, como os dados são protegidos e o que eu fiz e o que a IA fez.

## Os números

Tudo nesta tabela vem diretamente do repositório Git, do executor de testes e dos meus logs de sessão do Claude Code. As contagens de linhas deixam de fora as linhas em branco e o código gerado.

| O quê | Quantidade |
|---|---:|
| Dias corridos, do primeiro prompt à última versão | 12 (7 a 18 de outubro) |
| Tempo de sessão ativa com a IA | cerca de 20 horas |
| Minhas mensagens para a IA | cerca de 110 |
| Linhas de código não vazias (TypeScript, Swift, SQL, JavaScript) | 58.588 |
| Tabelas / migrações do banco de dados | 44 / 30 |
| Pull requests mesclados / versões com tag | 52 / 48 |
| Testes automatizados (Vitest, extensão do Chrome, pacote Swift) | 294 + 15 + 4 = 313 |
| Verificações no script de isolamento entre famílias | 34 |
| Triggers do banco de dados que garantem quem é dono do quê | 49 |
| Aplicações em um único backend | 7 |

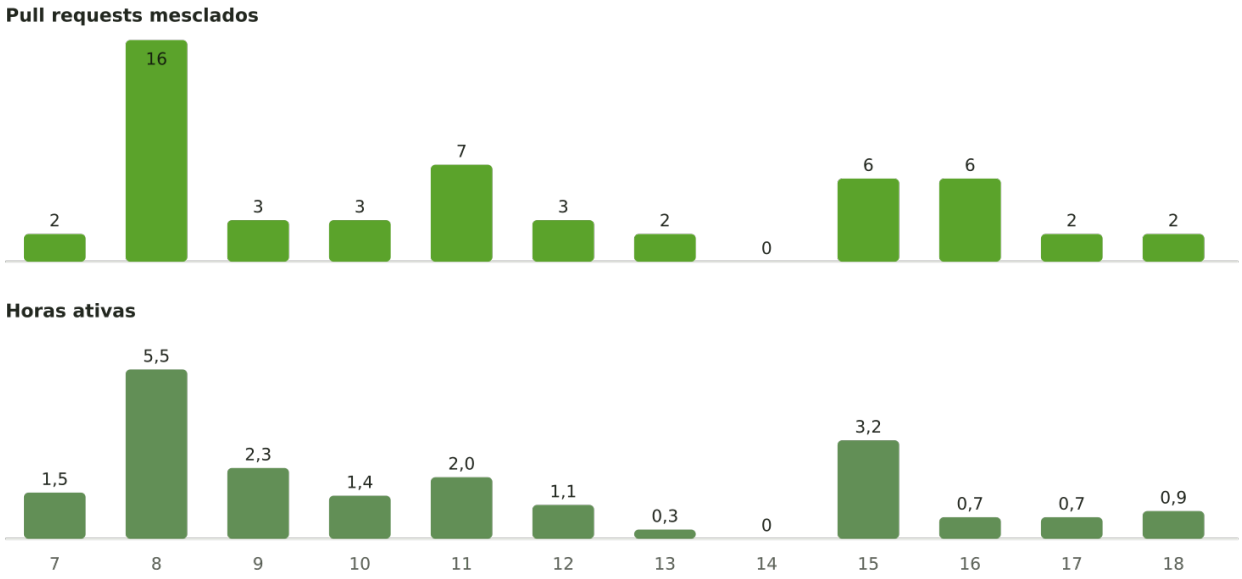
Veja como essas linhas se distribuem entre as sete aplicações e as suas partes compartilhadas. A aplicação web e o seu servidor carregam a maior parte do peso, como deve ser: é ali que vive cada regra, e todos os outros apps conversam com ela.




Linhas de código não vazias por parte do Hearth (58.588 no total)

O retrato dia a dia é mais interessante do que os totais, porque mostra o ritmo. Os pull requests vieram em rajadas, e as horas ativas se mantiveram modestas. O dia mais movimentado, 8 de outubro, teve dezesseis pull requests mesclados em cerca de cinco horas e meia de sessão -- do isolamento entre famílias e da Claude

API até uma lista de documentos mais tranquila --, depois que eu já tinha usado a primeira versão por um dia e sabia o que queria mudar.




Pull requests mesclados (em cima) e horas de sessão ativa (embaixo) por dia, de 7 a 18 de outubro

"Horas ativas" conta o tempo em que uma sessão estava ocupada, deixando de fora qualquer pausa de mais de vinte minutos. Em alguns momentos, várias sessões rodaram em paralelo, e essas sobreposições são contadas uma única vez.

> **O que os números não dizem**
> Vinte horas de sessão não são vinte horas da minha vida. Pensar, testar no iPad e na Apple TV e conviver com o app por um dia antes de decidir o que mudar, tudo isso aconteceu em volta delas. Linhas de código também não medem valor; Dijkstra argumentou que elas deveriam ser contabilizadas como "linhas gastas". Eu as uso aqui por um único motivo: elas dão uma noção de escala que os modelos clássicos de estimativa conseguem usar como comparação, e foi o que fiz no post anterior.

## Salvaguarda 1: toda mudança é um pull request

O vibe coding não tem histórico; você aceita, roda e segue em frente. O Hearth tem 52 pull requests mesclados, e cada um deles traz uma descrição por escrito do que mudou e por quê. Os maiores também dizem como a mudança foi verificada -- a descrição da revisão de segurança, por exemplo, termina com "tsc, biome, `pnpm build` e `pnpm check:isolation` (26/26) passam", além de um backup do banco de dados restaurado em um banco de rascunho e comparado com o de produção.

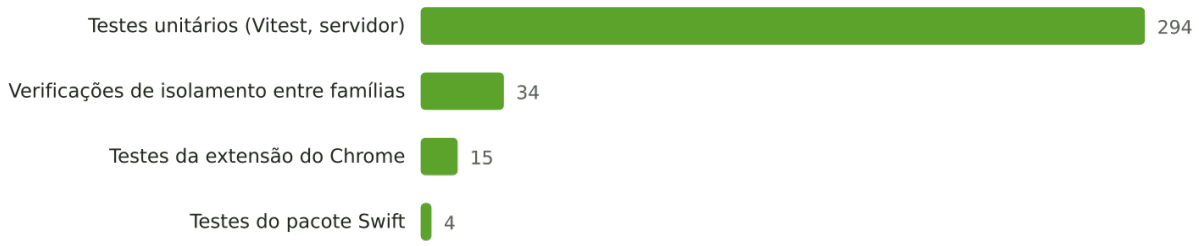
Pull requests pequenos não são burocracia. São eles que permitem encontrar a mudança que quebrou algo, ler um diff em cinco minutos em vez de uma hora e reverter exatamente uma coisa. O programa de pesquisa DORA diz há anos que lotes pequenos e testes completos fazem diferença, e o relatório de 2024 constatou que a adoção de IA andou junto com uma estabilidade de entrega *menor*. Na minha opinião, essa constatação não é um argumento contra a IA. É um argumento para manter os lotes pequenos justamente quando a IA torna tentador fazê-los enormes.

## Salvaguarda 2: testes que encontraram bugs reais no primeiro dia

Em 12 de outubro, o projeto ganhou os seus testes unitários: 186 no início, 294 hoje, escritos com Vitest para as funções puras em `src/lib` -- cálculos de datas, agendamentos, regras de negócio, parsers e guardas de requisição. Eles rodam em cerca de meio segundo, então ninguém tem motivo para pulá-los. Um detalhe de que eu gosto bastante: a máquina de testes finge estar em Honolulu, que não é nem UTC (o servidor) nem Nova York (a casa). Código que se esquece de informar um fuso horário falha ali, e não em produção.

A primeira execução encontrou quatro bugs reais, cada um corrigido com um teste que falhou primeiro.

1. **Orçamentos no dia errado.** Dois orçamentos enviados às 15h e às 21h30, horário de Nova York, caíram em "rodadas" diferentes, porque 21h30 já é o dia seguinte em UTC. O orçamento mais barato, o da tarde, contou como substituído.
2. **Um open redirect.** A função que protege o "voltar a esta página depois do login" aceitava `/\t/evil.example`. Os navegadores descartam tabulações e quebras de linha, então esse endereço levava a outro site.
3. **Nomes herdados.** Um parser aceitava nomes que todo objeto JavaScript herda, como `toString`.
4. **Dias que não existem.** A validação de datas aceitava `2026-02-30`.



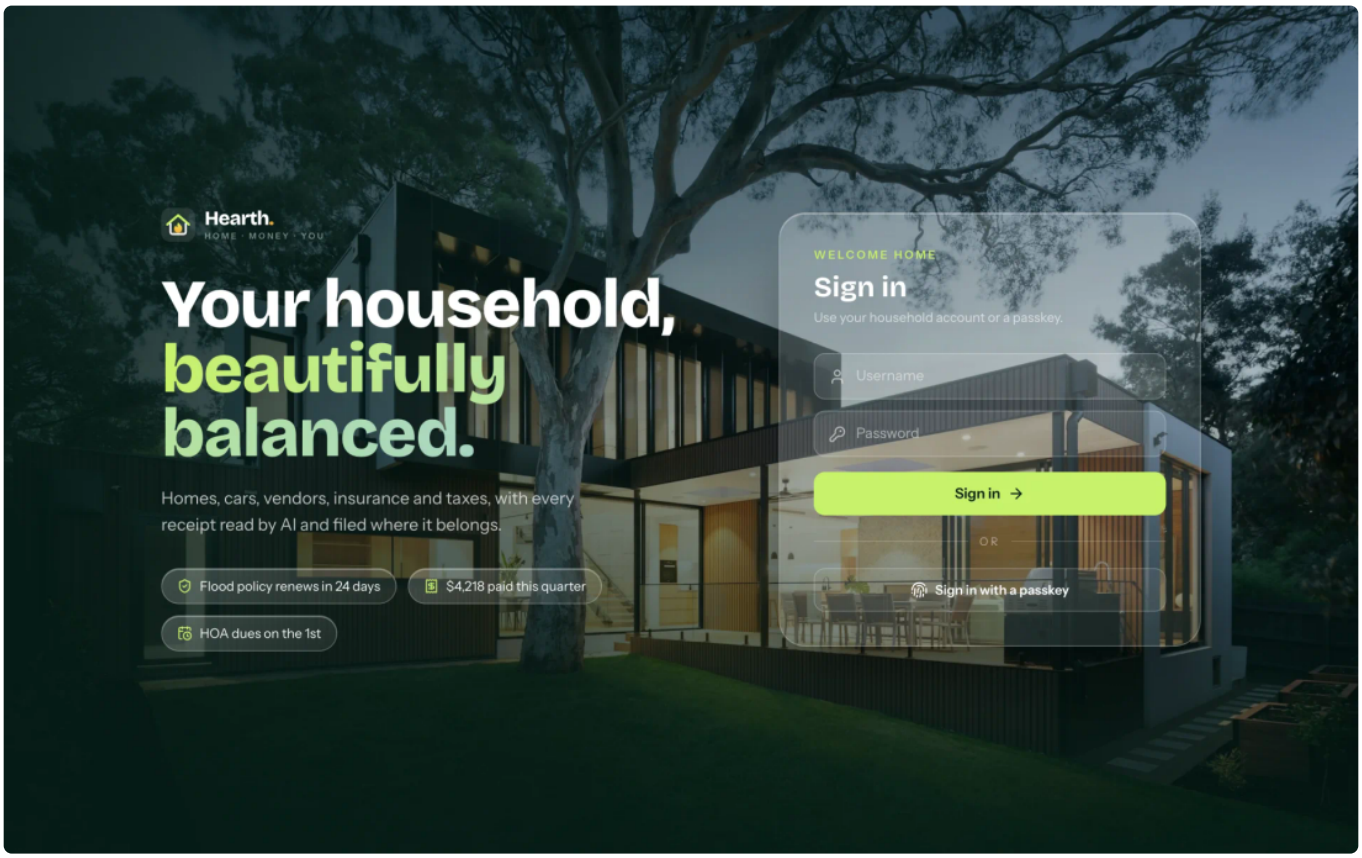

Verificações automatizadas no Hearth: testes unitários, verificações de isolamento, testes da extensão e do Swift

No mesmo dia, os testes viraram lei. O script de deploy os executa antes de construir qualquer coisa e se recusa a publicar quando algum falha, não importa quem faça o deploy. Para provar que a barreira realmente funciona, quebramos a função `relativeDays` de propósito e vimos o deploy parar com "Unit tests failed: not deploying." Depois, corrigimos de novo. As instruções de release também proíbem pular, apagar ou afrouxar um teste para fazer uma versão passar -- uma regra escrita tanto para a IA quanto para mim.

> **Onde os testes param**
> Os testes unitários cobrem lógica, não telas. Ainda não há testes end-to-end no navegador, e os apps em Swift têm só um punhado de testes no pacote compartilhado. Cada mudança de tela foi verificada rodando o app -- no navegador, nos simuladores, no iPad de verdade -- e os pull requests dizem isso, mas isso é verificação manual, não automação. Para um produto com muitos clientes, em vez de uma única casa, testes end-to-end seriam a próxima coisa que eu acrescentaria.

## Salvaguarda 3: login, do jeito certo

O login é um bom lugar para distinguir trabalho profissional de uma demo, porque uma demo para em "ele me deixa entrar". O Hearth oferece senhas e passkeys, o padrão de login resistente a phishing da FIDO Alliance, construído sobre o WebAuthn do W3C (implementado com SimpleWebAuthn). As senhas são armazenadas com bcrypt; um nome de usuário desconhecido é verificado contra um hash fictício de mesmo custo, então a resposta leva exatamente o mesmo tempo que para um usuário real e não revela quais contas existem.




A página de login: senha ou passkey, com limitação de tentativas por trás das duas (instância de demonstração).

É nos detalhes que está o trabalho. Tentativas que falham são limitadas por nome de usuário e por endereço, e os contadores ficam no banco de dados, para que uma reinicialização não os zere. Os tokens de sessão são aleatórios e armazenados apenas como HMAC, então uma cópia do banco de dados não faz login de ninguém, e o cookie é `httpOnly`, `secure` e `sameSite`. Cada route handler verifica a origem da requisição. Trocar a senha encerra as suas outras sessões e todos os apps conectados, adicionar uma passkey pede a senha primeiro, e quando um administrador redefine uma senha, as passkeys também são removidas. E cada server action e cada rota de API verifica o usuário por conta própria, em vez de confiar que algum middleware já fez isso.

## Salvaguarda 4: uma família nunca vê os dados de outra

O Hearth é multi-tenant: várias famílias e, dentro de uma família, registros compartilhados e registros privados de um único membro. Um erro aqui não é um bug; é um incidente de privacidade. Por isso, o isolamento é garantido três vezes, em três níveis diferentes.

Três camadas independentes mantêm os registros de uma família longe das outras

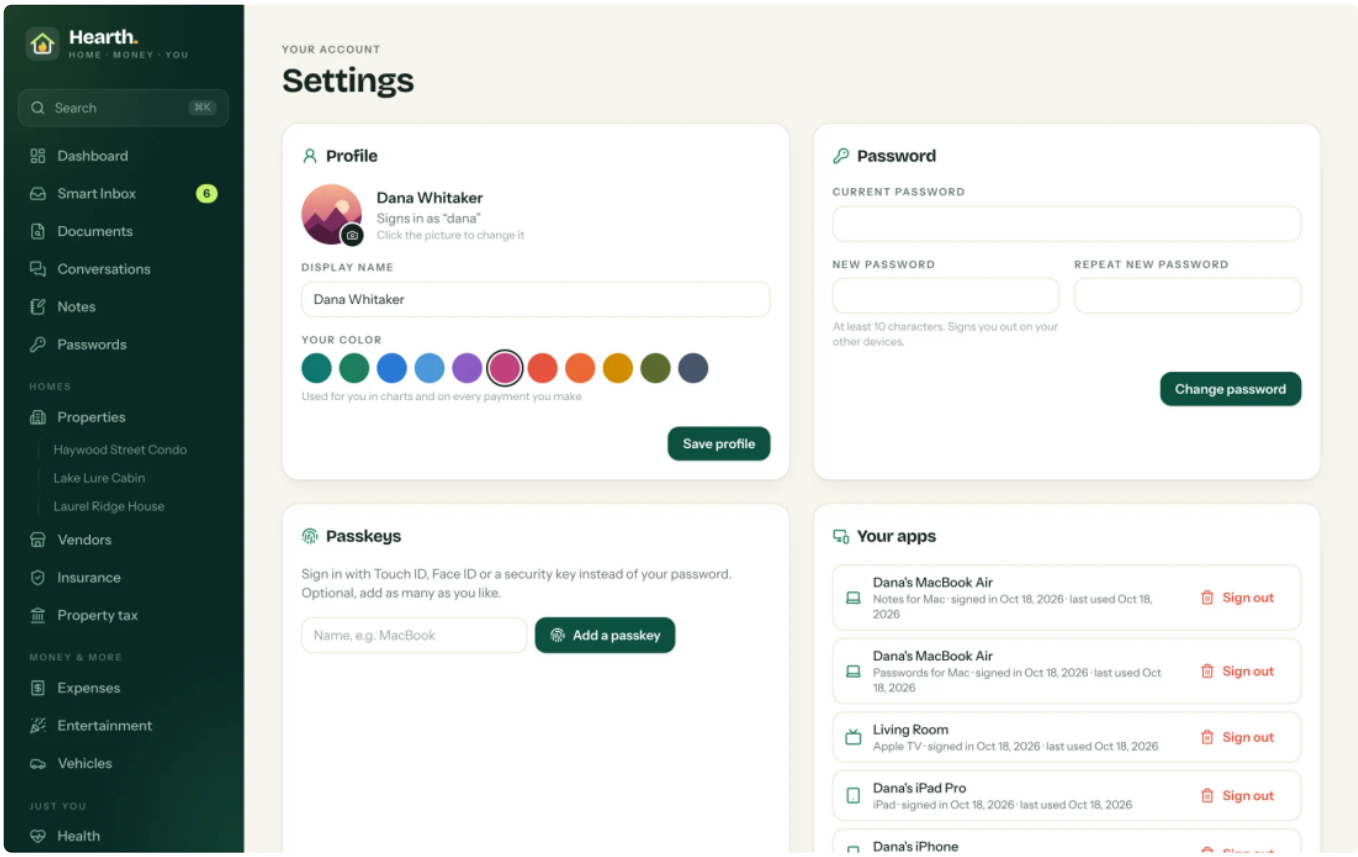
A camada interna é um cliente de banco de dados que acrescenta automaticamente as regras de família e de membro a cada consulta, inclusive em relações aninhadas e contagens, para que um desenvolvedor -- ou uma IA -- não possa esquecê-las. A camada do meio não confia na interna: quarenta e nove triggers no PostgreSQL verificam a posse dos dados quando uma transação é confirmada. A camada externa não confia em nenhuma das duas: um script cria uma família descartável, tenta 34 formas de ler, alterar ou vincular registros além da fronteira, e depois a remove de novo.

Na minha opinião, essa é a diferença mais importante entre um protótipo e um produto. Qualquer um consegue escrever `where familyId = ?`. A questão é o que acontece no dia em que alguém esquece.

## Salvaguarda 5: segredos continuam secretos

O cofre de senhas criptografa cada campo de uma entrada -- título, endereço, nome de usuário, senha e notas -- com AES-256-GCM, usando uma chave derivada de um segredo do servidor. Cada valor fica vinculado à sua família, à sua entrada e ao seu campo como dados autenticados adicionais. Copie uma senha criptografada de uma linha para outra, e ela não abre mais. As senhas anteriores ficam em um histórico, criptografadas do mesmo jeito, então uma senha antiga nunca se perde.

Os apps seguem a mesma ideia de privilégio mínimo. Cada um faz login separadamente, com um código confirmado na web, e o seu token é armazenado apenas como HMAC. Cada tipo de app só pode fazer aquilo para que foi feito: a Apple TV e os apps para iPhone e iPad só leem e nunca alteram nada, o Notes for Mac só vê notas e a extensão do Chrome só vê senhas. Dentro da extensão, apenas o background worker guarda o token, e ele só responde a uma página web sobre o próprio site daquela página, depois de um clique de verdade.




Settings ' Your apps: cada app conectado, com o seu tipo, quando fez login e quando foi usado pela última vez -- cada um pode ser desconectado individualmente.

## Salvaguarda 6: uma revisão que encontrou o que a IA errou

Em 11 de outubro, pedi à IA que revisasse a sua própria base de código quanto a autorização, privacidade entre membros e segurança -- não como formalidade, mas com a tarefa explícita de encontrar problemas. Ela encontrou problemas reais, e é justamente para isso que se faz uma revisão. Entre eles havia uma falha de **stored cross-site scripting** (XSS armazenado): um upload podia escolher o próprio content type, e um arquivo forjado poderia ter executado script no app. Ela também encontrou pontos em que editar um registro privado podia torná-lo visível para o resto da família, e um relatório de uso de IA que mostrava títulos de documentos ao administrador.

Tudo isso foi corrigido no mesmo pull request, junto com a limitação de tentativas no banco de dados, o encerramento das sessões ao trocar a senha e os backups noturnos. Quero ser honesto sobre o que isso significa. A IA escreveu esses bugs, e a IA os encontrou -- porque eu pedi uma revisão e a tratei como uma barreira, não como um ritual. Isso não é diferente de como uma boa equipe trabalha com desenvolvedores júnior. Ninguém publica nada sem um segundo par de olhos.

## Salvaguarda 7: versões e operação

O Hearth roda em um Raspberry Pi 5 em casa, em um contêiner Docker atrás do Caddy, que cuida dos certificados HTTPS. Toda versão segue o mesmo pipeline, e o pipeline está escrito como uma skill que a IA executa passo a passo, então uma versão publicada numa noite de cansaço fica exatamente igual a uma publicada numa manhã descansada.

Do pull request à produção: toda versão segue o mesmo caminho

As últimas cinco imagens de versão ficam no Pi, então um rollback é uma linha no arquivo compose. O banco de dados é exportado toda noite para um bucket de armazenamento separado, com cópias diárias e mensais, e um backup foi restaurado em um banco de rascunho e comparado com o de produção antes que eu confiasse nele. Aprendi essa lição do jeito difícil, como descrevi em My Server Was Compromised: a arquitetura decide o raio de impacto muito antes de qualquer coisa dar errado.

## O que a IA fez, e o que eu fiz

A divisão de trabalho é a parte em que as pessoas têm mais dificuldade de acreditar, então vou ser concreto. Em doze dias, escrevi cerca de 110 mensagens; a IA fez aproximadamente 3.900 chamadas de ferramentas em resposta -- lendo arquivos, escrevendo código, rodando testes, fazendo build, fazendo deploy. Ela escreveu quase todo o código, os testes e as descrições dos pull requests.

O que ela não fez foi decidir. Eu escolhi a arquitetura e a stack: Next.js 16 com React 19, Prisma 7 sobre PostgreSQL, TypeScript 7 com o seu novo compilador nativo, SwiftUI para os apps da Apple e uma extensão do Chrome em Manifest V3. Decidi que o isolamento é garantido no banco de dados, e não só no código, que as senhas ficam vinculadas à sua linha, que cada app recebe só a sua parte e que os testes barram cada versão. Testei em dispositivos reais, anotei as regras que o app deveria seguir ("datas primeiro, mais recentes no topo", "nada de dropdowns simples, sempre com busca") e disse não -- com frequência. Quando o login do iPad ficou parecido com a tela de código da Apple TV, foi uma pessoa que percebeu.

> **A pesquisa diz: depende do desenvolvedor**
> Não quero fingir que as evidências apontam para um lado só. Em um ensaio randomizado controlado do METR, dezesseis desenvolvedores experientes de código aberto foram 19% mais lentos com as ferramentas de IA do início de 2025 em suas próprias bases de código grandes, enquanto acreditavam estar mais rápidos. O próprio METR aponta os limites -- uma amostra pequena, um único tipo de trabalho, ferramentas que mudaram desde então -- e já publicou dados mais recentes. A lição que eu tiro não é que a IA seja lenta ou rápida. É que o ganho depende de quem a conduz: de quão precisamente essa pessoa especifica, de quão rigorosamente revisa e de ela tratar ou não testes e revisões como barreiras. Sem isso, você fica com vibes. Com isso, você chega ao Hearth.

## O que isso prova

Rápido e cuidadoso já não são opostos. A mesma IA que escreveu 58.588 linhas em doze dias também escreveu os 313 testes, as 34 verificações de isolamento, os 49 triggers e a revisão de segurança que mantiveram essas linhas honestas -- porque foi isso que pedi a ela, e porque nada chegou à produção sem passar por eles. A velocidade veio das ferramentas. O cuidado veio de trinta anos sabendo o que dá errado em produção, e de me recusar a pular as partes chatas.

> A IA não eliminou a engenharia. Eliminou a desculpa para pulá-la.

Se você quer esse tipo de velocidade para o seu próprio produto -- uma aplicação Delphi que precisa de um futuro moderno, ou um sistema que se encaixe no seu negócio, e não o contrário --, dê uma olhada no que eu construí, no que eu ofereço e depois vamos conversar.

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 —**