NEWS · OCTOBER 20, 2026

# Kein Vibe Coding: 58.588 Zeilen, 313 Tests und 48 Releases in 12 Tagen

Zwölf Tage, 58.588 Zeilen, 313 automatisierte Tests, 34 Isolationsprüfungen, 52 Pull-Requests und 48 Releases -- die Zahlen und die Leitplanken hinter Hearth, und warum schnelle KI-gestützte Entwicklung und sorgfältiges Engineering keine Gegensätze sind.

By Dr. Holger Flick

Wann immer ich zeige, was ich mit KI baue, stellt jemand die Frage höflich und jemand anderes etwas weniger höflich: „Ist das nicht einfach Vibe Coding?“ Die Frage ist berechtigt. Der Begriff stammt von Andrej Karpathy, der im Februar 2025 eine Art zu programmieren beschrieb, bei der man „sich ganz den Vibes hingibt, die Exponentialkurven umarmt und vergisst, dass der Code überhaupt existiert“ -- und jede Änderung übernimmt, ohne die Diffs zu lesen (Wikipedia sammelt den ursprünglichen Beitrag und die Debatte seither). Für einen

Prototyp am Wochenende ist das eine völlig vernünftige Art, Spaß zu haben.

Für Software, die die Versicherungspolicen, Arztrechnungen und Passwörter einer Familie verwahrt, ist es das nicht. In meinem vorigen Beitrag habe ich Hearth vorgestellt, das Haushaltssystem, das ich in zwölf Tagen für

meine Frau gebaut habe. Dieser Beitrag ist die andere Hälfte der Geschichte: die Zahlen und die Leitplanken, die sie hervorgebracht haben. Meine These ist einfach, und ich belege sie mit dem Repository selbst. Schnell und sorgfältig sind keine Gegensätze -- KI hat auch die sorgfältigen Teile billiger gemacht.

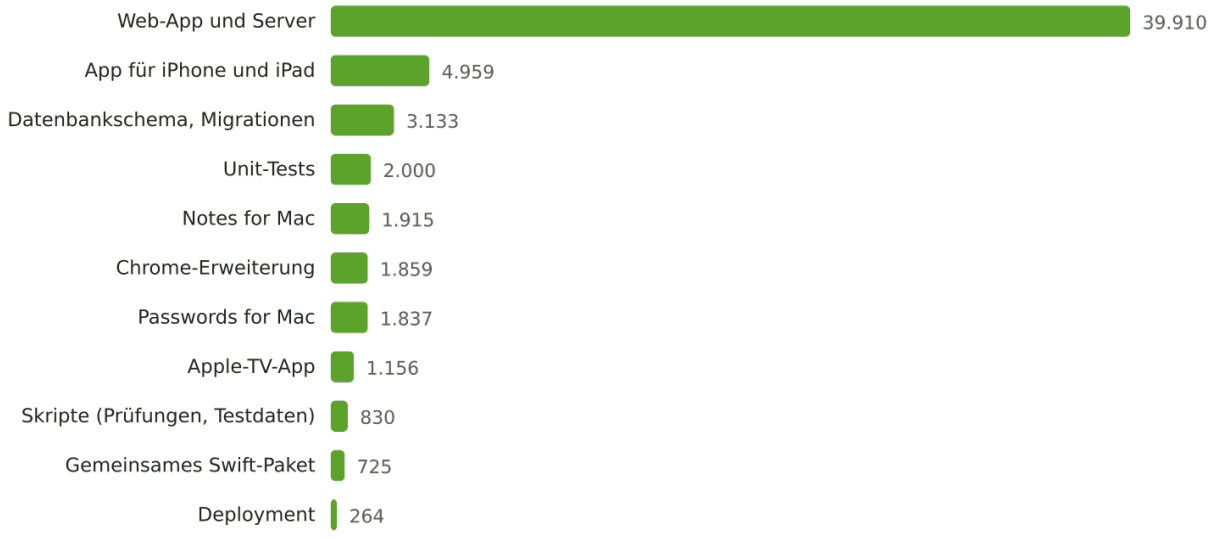
Am Ende wissen Sie genau, was gemessen wurde, wie die Tests und Reviews aufgesetzt waren, wie die Daten geschützt sind und was ich getan habe und was die KI.

# Die Zahlen

Alles in dieser Tabelle stammt direkt aus dem Git-Repository, dem Test-Runner und meinen Claude-Code-Sitzungsprotokollen. Bei den Zeilenzahlen sind Leerzeilen und generierter Code nicht mitgezählt.

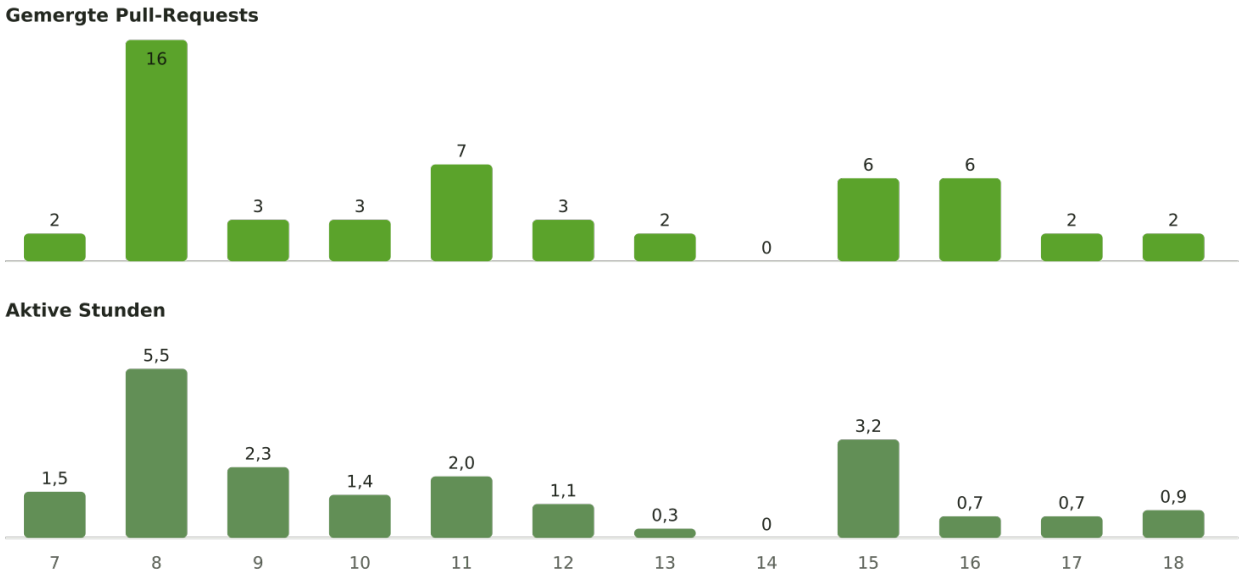
| Was | Anzahl |
|---|---:|
| Kalendertage, vom ersten Prompt bis zum letzten Release | 12 (7. bis 18. Oktober) |
| Aktive Sitzungszeit mit der KI | etwa 20 Stunden |
| Meine Nachrichten an die KI | etwa 110 |
| Nicht leere Codezeilen (TypeScript, Swift, SQL, JavaScript) | 58.588 |
| Datenbanktabellen / Migrationen | 44 / 30 |
| Gemergte Pull-Requests / getaggte Releases | 52 / 48 |
| Automatisierte Tests (Vitest, Chrome-Erweiterung, Swift-Paket) | 294 + 15 + 4 = 313 |
| Prüfungen im Skript für die Familien-Isolation | 34 |
| Datenbank-Trigger, die durchsetzen, wem was gehört | 49 |
| Anwendungen auf einem Backend | 7 |

So verteilen sich diese Zeilen auf die sieben Anwendungen und ihre gemeinsamen Teile. Die Webanwendung und ihr Server tragen den Großteil des Gewichts, und so soll es sein: Dort liegt jede Regel, und jede andere App spricht mit ihr.




Nicht leere Codezeilen je Teil von Hearth (insgesamt 58.588)

Der Verlauf Tag für Tag ist interessanter als die Summen, weil er den Rhythmus zeigt. Die Pull-Requests kamen in Schüben, und die aktiven Stunden blieben überschaubar. Am arbeitsreichsten Tag, dem 8. Oktober, wurden sechzehn Pull-Requests in etwa fünfeinhalb Stunden Sitzungszeit gemergt -- von der Familien-Isolation und der Claude API bis zu einer ruhigeren Dokumentenliste --, nachdem ich die erste Version einen Tag lang benutzt hatte und wusste, was ich geändert haben wollte.




Gemergte Pull-Requests (oben) und aktive Sitzungsstunden (unten) pro Tag, 7. bis 18. Oktober

„Aktive Stunden“ zählt die Zeit, in der eine Sitzung beschäftigt war, ohne Pausen über zwanzig Minuten. Zeitweise liefen mehrere Sitzungen parallel; diese Überschneidungen sind nur einmal gezählt.

> **Was die Zahlen nicht sagen**
>
> Zwanzig Stunden Sitzungszeit sind nicht zwanzig Stunden meines Lebens. Nachdenken, Testen auf dem iPad und dem Apple TV und das Leben mit der App für einen Tag, bevor ich entschied, was sich ändern sollte, passierten alles drumherum. Codezeilen messen auch keinen Wert; Dijkstra hat dafür plädiert, sie als „verbrauchte Zeilen“ zu verbuchen. Ich verwende sie hier nur aus einem Grund: Sie vermitteln eine Vorstellung von der Größenordnung, mit der sich die klassischen Schätzmodelle vergleichen lassen, und genau das habe ich im vorigen Beitrag getan.

## Leitplanke 1: Jede Änderung ist ein Pull-Request

Vibe Coding hat keine Geschichte; man übernimmt, man startet, man macht weiter. Hearth hat 52 gemergte Pull-Requests, und jeder davon trägt eine schriftliche Beschreibung dessen, was sich geändert hat und warum. Die größeren sagen außerdem, wie die Änderung überprüft wurde -- die Beschreibung des Sicherheits-Reviews etwa endet mit „tsc, biome, `pnpm build` and `pnpm check:isolation` (26/26) pass“, dazu ein Datenbank-Backup, das in eine Wegwerf-Datenbank zurückgespielt und mit der Live-Datenbank verglichen wurde.

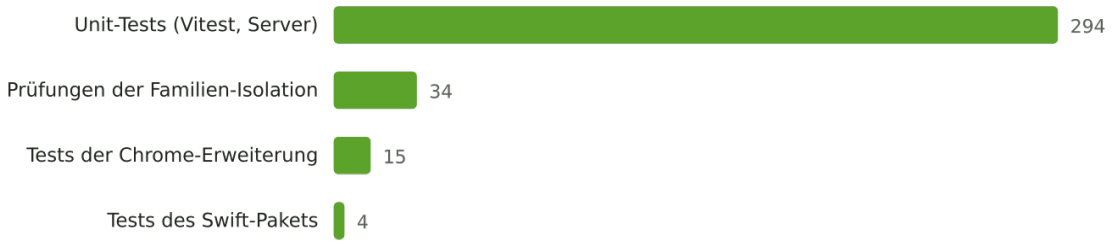
Kleine Pull-Requests sind keine Bürokratie. Sie sorgen dafür, dass man die Änderung findet, die etwas kaputt gemacht hat, dass man einen Diff in fünf Minuten statt in einer Stunde liest und dass man genau eine Sache zurückrollen kann. Das DORA-Forschungsprogramm sagt seit Jahren, dass kleine Batches und gründliches Testen wichtig sind, und sein Report von 2024 stellte fest, dass der Einsatz von KI mit einer *geringeren* Stabilität der Auslieferung einherging. Meiner Meinung nach ist dieser Befund kein Argument gegen KI. Er ist ein Argument dafür, die Batches klein zu halten, wenn die KI es verlockend macht, sie riesig werden zu lassen.

## Leitplanke 2: Tests, die am ersten Tag echte Bugs fanden

Am 12. Oktober bekam das Projekt seine Unit-Tests: anfangs 186, heute 294, geschrieben mit Vitest für die reinen Funktionen in `src/lib` -- Datumsrechnung, Zeitpläne, Geschäftsregeln, Parser und Request-Guards. Sie laufen in etwa einer halben Sekunde, also hat niemand einen Grund, sie zu überspringen. Ein Detail, das mir ziemlich gefällt: Die Testmaschine gibt vor, in Honolulu zu stehen, und das ist weder UTC (der Server) noch New York (der Haushalt). Code, der vergisst, eine Zeitzone anzugeben, scheitert dort statt in der Produktion.

Der erste Lauf fand vier echte Bugs, und jeder wurde mit einem Test behoben, der zuerst fehlschlug.

1. **Angebote am falschen Tag.** Zwei Angebote, um 15 Uhr und um 21:30 Uhr New Yorker Zeit verschickt, landeten in verschiedenen „Runden“, weil 21:30 Uhr in UTC schon der nächste Tag ist. Das günstigere Angebot vom Nachmittag galt als ersetzt.
2. **Ein Open Redirect.** Die Funktion, die „nach der Anmeldung auf diese Seite zurückkehren“ absichert, akzeptierte `/\t/evil.example`. Browser verwerfen Tabulatoren und Zeilenumbrüche, also führte diese Adresse auf eine fremde Website.
3. **Geerbte Namen.** Ein Parser akzeptierte Namen, die jedes JavaScript-Objekt erbt, etwa `toString`.
4. **Tage, die es nicht gibt.** Die Datumsprüfung akzeptierte `2026-02-30`.



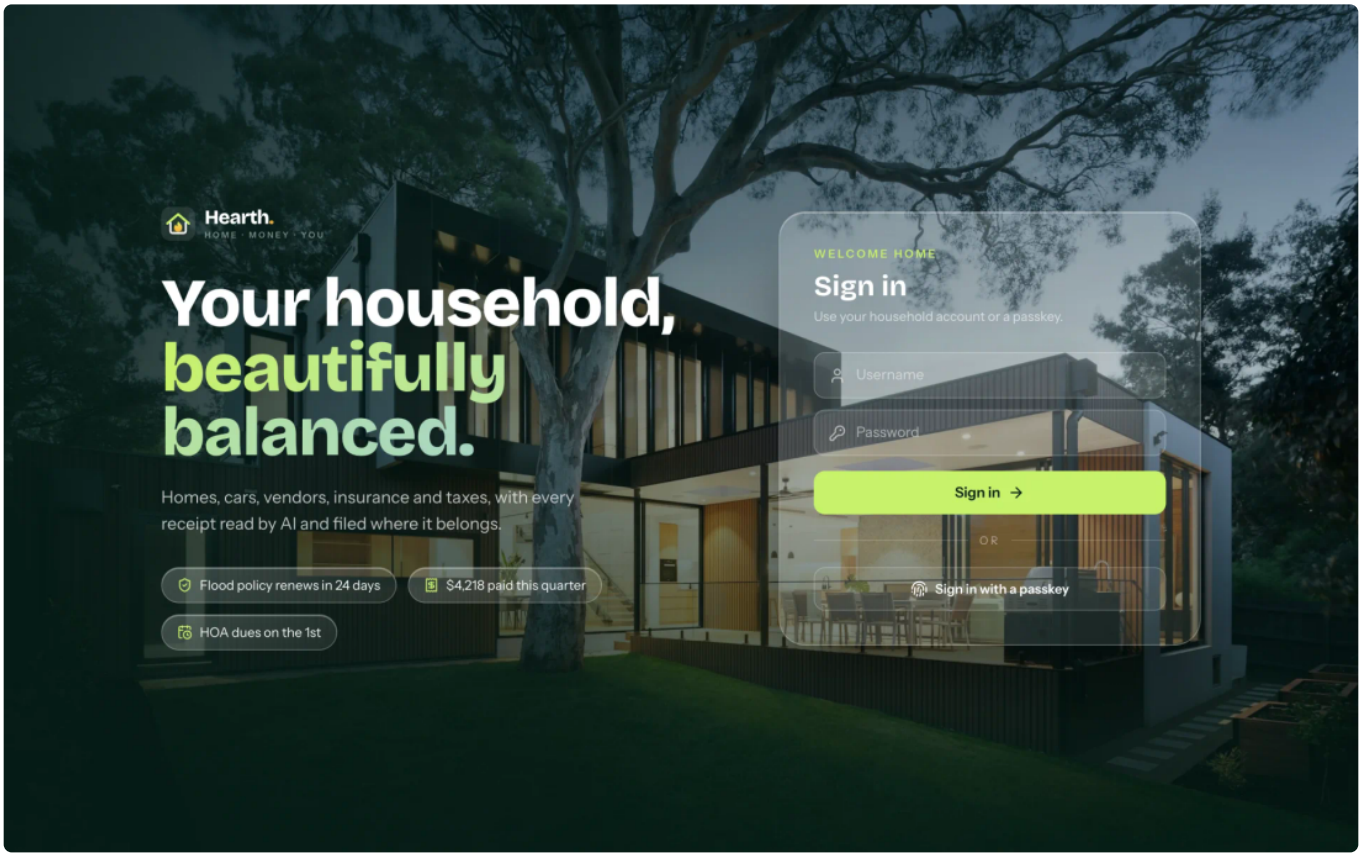

Automatisierte Prüfungen in Hearth: Unit-Tests, Isolationsprüfungen, Tests der Erweiterung und des Swift-Pakets

Am selben Tag wurden die Tests zum Gesetz. Das Deploy-Skript führt sie aus, bevor es irgendetwas baut, und verweigert die Auslieferung, wenn einer fehlschlägt, ganz gleich, wer deployt. Um zu beweisen, dass die Schranke wirklich funktioniert, haben wir die Funktion `relativeDays` absichtlich kaputt gemacht und zugesehen, wie das Deployment mit „Unit tests failed: not deploying.“ abbrach. Danach haben wir sie wieder repariert. Die Release-Anweisungen verbieten außerdem, einen Test zu überspringen, zu löschen oder aufzuweichen, um ein Release durchzubringen -- eine Regel, die ebenso für die KI geschrieben ist wie für mich.

> **Wo die Tests aufhören**
> Die Unit-Tests decken Logik ab, keine Oberflächen. Browserbasierte End-to-End-Tests gibt es noch nicht, und die Swift-Apps haben nur eine Handvoll Tests in ihrem gemeinsamen Paket. Jede Änderung an einer Oberfläche wurde geprüft, indem die App ausgeführt wurde -- im Browser, in den Simulatoren, auf dem echten iPad --, und die Pull-Requests sagen das auch, aber das ist manuelle Prüfung, keine Automatisierung. Für ein Produkt mit vielen Kunden statt einem Haushalt wären End-to-End-Tests das Nächste, was ich ergänzen würde.

## Leitplanke 3: Anmeldung, richtig gemacht

An der Anmeldung lässt sich professionelle Arbeit gut von einer Demo unterscheiden, denn eine Demo hört bei „es lässt mich rein“ auf. Hearth bietet Passwörter und Passkeys, den phishing-resistenten Anmeldestandard der FIDO Alliance, der auf WebAuthn des W3C aufbaut (umgesetzt mit SimpleWebAuthn). Passwörter werden mit bcrypt gespeichert; ein unbekannter Benutzername wird gegen einen Dummy-Hash mit demselben Aufwand geprüft, sodass die Antwort genauso lange dauert wie bei einem echten und nicht verrät, welche Konten existieren.

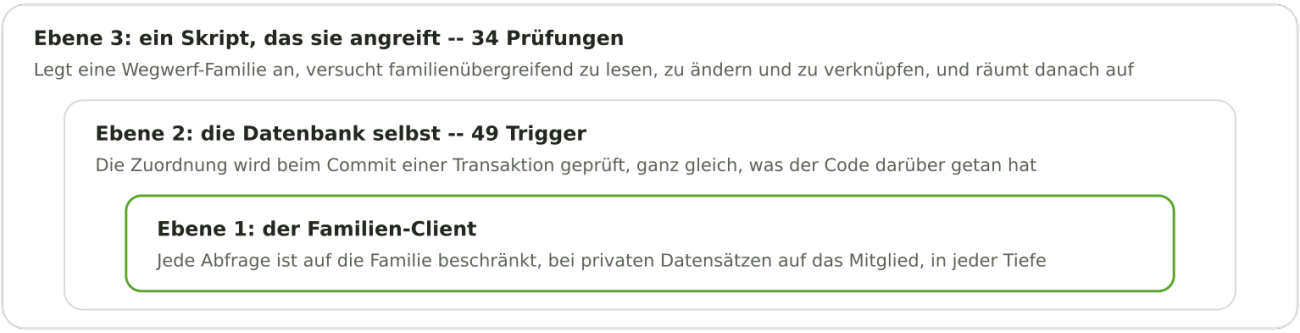



Die Anmeldeseite: Passwort oder Passkey, mit Drosselung hinter beidem (Demo-Instanz).

Die Arbeit steckt in den Details. Fehlgeschlagene Versuche werden pro Benutzername und pro Adresse gedrosselt, und die Zähler liegen in der Datenbank, damit ein Neustart sie nicht zurücksetzt. Sitzungstoken sind zufällig und nur als HMAC gespeichert, sodass eine Kopie der Datenbank niemanden anmeldet, und das Cookie ist `httpOnly`, `secure` und `sameSite`. Jeder Route-Handler prüft die Herkunft der Anfrage. Wer sein Passwort ändert, meldet damit seine anderen Sitzungen und jede verbundene App ab, das Hinzufügen eines Passkeys verlangt zuerst das Passwort, und wenn ein Administrator ein Passwort zurücksetzt, werden auch die Passkeys entfernt. Und jede Server-Action und jede API-Route prüft den Benutzer selbst, statt darauf zu vertrauen, dass irgendeine Middleware das schon erledigt hat.

## Leitplanke 4: Eine Familie sieht niemals die Daten einer anderen

Hearth ist mandantenfähig: mehrere Familien und innerhalb einer Familie Datensätze, die geteilt werden, und solche, die nur einem Mitglied gehören. Ein Fehler ist hier kein Bug, sondern ein Datenschutzvorfall. Deshalb wird die Isolation dreimal durchgesetzt, auf drei verschiedenen Ebenen.




Drei unabhängige Ebenen halten die Datensätze einer Familie von einer anderen fern

Die innere Ebene ist ein Datenbank-Client, der jeder Abfrage automatisch die Regeln für Familie und Mitglied hinzufügt, auch verschachtelten Relationen und Zählungen, sodass ein Entwickler -- oder eine KI -- sie nicht vergessen kann. Die mittlere Ebene traut der inneren nicht: Neunundvierzig Trigger in PostgreSQL prüfen die

Zugehörigkeit, wenn eine Transaktion committet wird. Die äußere Ebene traut keiner von beiden: Ein Skript legt eine Wegwerf-Familie an, probiert 34 Wege aus, über die Grenze hinweg zu lesen, zu ändern oder zu verknüpfen, und entfernt sie danach wieder.

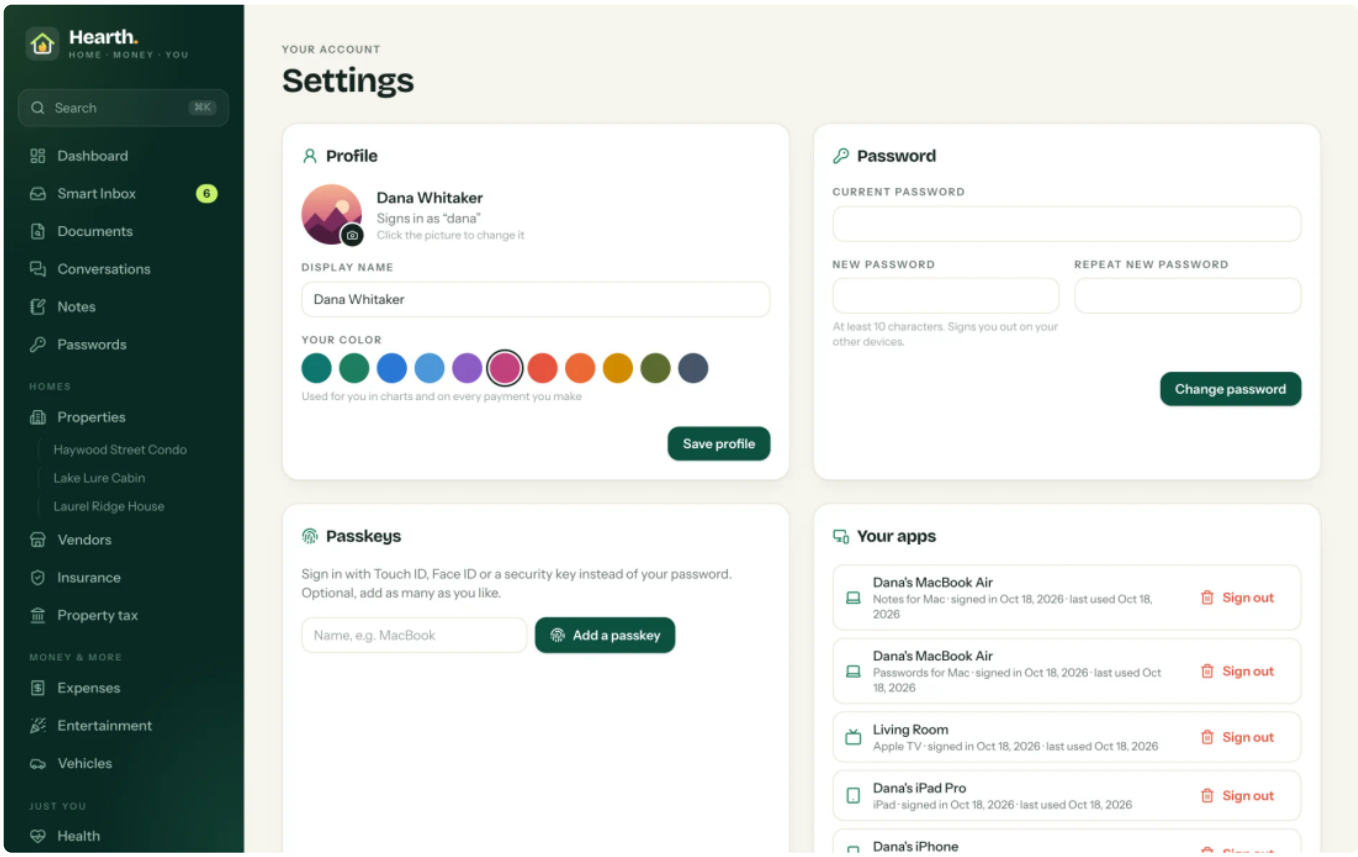
Meiner Meinung nach ist das der wichtigste Unterschied zwischen einem Prototyp und einem Produkt. `where familyId = ?` kann jeder schreiben. Die Frage ist, was an dem Tag passiert, an dem es jemand vergisst.

## Leitplanke 5: Geheimnisse bleiben geheim

Der Passwort-Tresor verschlüsselt jedes Feld eines Eintrags -- Titel, Adresse, Benutzername, Passwort und Notizen -- mit AES-256-GCM, mit einem Schlüssel, der aus einem Server-Geheimnis abgeleitet wird. Jeder

Wert ist als zusätzliche authentifizierte Daten (AAD) an seine Familie, seinen Eintrag und sein Feld gebunden. Kopiert man ein verschlüsseltes Passwort aus einer Zeile in eine andere, lässt es sich nicht mehr öffnen. Frühere Passwörter bleiben in einem Verlauf erhalten, auf dieselbe Weise verschlüsselt, sodass ein altes Passwort nie verloren geht.

Die Apps folgen demselben Prinzip der minimalen Rechte. Jede meldet sich einzeln mit einem Code an, der im Web bestätigt wird, und ihr Token ist nur als HMAC gespeichert. Jede Art von App darf nur das tun, wofür sie gemacht ist: Apple TV und die Apps für iPhone und iPad lesen nur und ändern nie etwas, Notes for Mac sieht nur Notizen, und die Chrome-Erweiterung sieht nur Passwörter. Innerhalb der Erweiterung hält nur der Hintergrund-Worker das Token, und er antwortet einer Webseite nur zu deren eigener Website, und das erst nach einem echten Klick.




Einstellungen ' Ihre Apps: jede angemeldete App mit ihrer Art, dem Zeitpunkt der Anmeldung und der letzten Nutzung -- jede lässt sich einzeln abmelden.

## Leitplanke 6: Ein Review, das fand, was die KI falsch gemacht hatte

Am 11. Oktober ließ ich die KI ihre eigene Codebasis auf Autorisierung, Privatsphäre zwischen den Mitgliedern und Sicherheit prüfen -- nicht als Formalität, sondern mit dem ausdrücklichen Auftrag, Probleme zu finden. Sie fand echte, und genau darum macht man das. Darunter war eine Lücke für **Stored Cross-Site Scripting**:

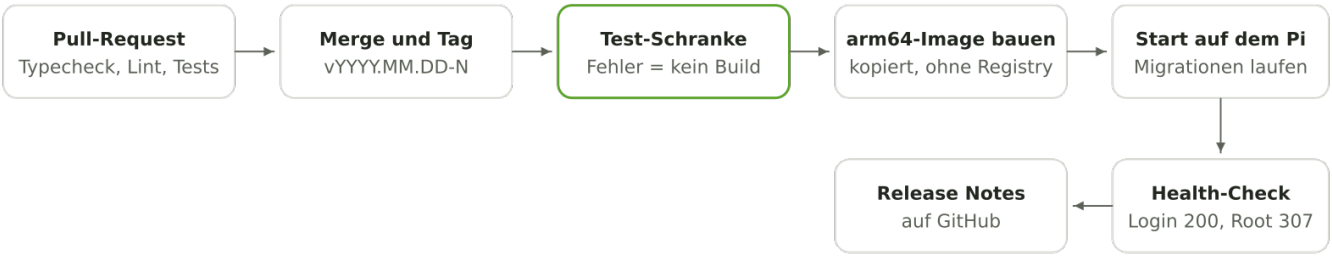
Ein Upload konnte seinen eigenen Content-Type bestimmen, und eine präparierte Datei hätte Skripte in der App ausführen können. Außerdem fand sie Stellen, an denen das Bearbeiten eines privaten Datensatzes ihn für den Rest der Familie sichtbar machen konnte, und einen Bericht zur KI-Nutzung, der dem Administrator Dokumenttitel anzeigte.

All das wurde im selben Pull-Request behoben, zusammen mit der Drosselung in der Datenbank, den Abmeldungen bei einer Passwortänderung und nächtlichen Backups. Ich möchte ehrlich sagen, was das bedeutet. Die KI hat diese Bugs geschrieben, und die KI hat sie gefunden -- weil ich um ein Review gebeten und es als Schranke behandelt habe, nicht als Ritual. Das ist nicht anders, als ein gutes Team mit Junior-Entwicklern arbeitet. Niemand liefert aus, ohne dass ein zweites Paar Augen draufgeschaut hat.

## Leitplanke 7: Releases und Betrieb

Hearth läuft auf einem Raspberry Pi 5 im Haus, in einem Docker-Container hinter Caddy, das sich um

die HTTPS-Zertifikate kümmert. Jedes Release folgt derselben Pipeline, und diese Pipeline ist als Skill aufgeschrieben, den die KI Schritt für Schritt ausführt, sodass ein Release an einem müden Abend genauso aussieht wie eines an einem frischen Morgen.




Vom Pull-Request in die Produktion: Jedes Release nimmt denselben Weg

Die letzten fünf Release-Images bleiben auf dem Pi, sodass ein Rollback eine einzige Zeile in der Compose-Datei ist. Die Datenbank wird jede Nacht in einen separaten Storage-Bucket gesichert, mit täglichen und monatlichen Kopien, und ein Backup wurde in eine Wegwerf-Datenbank zurückgespielt und mit der Live-Datenbank verglichen, bevor ich ihm vertraut habe. Diese Lektion habe ich auf die harte Tour gelernt, wie ich in Mein Server wurde kompromittiert beschrieben habe: Die Architektur entscheidet über den Explosionsradius, lange bevor irgendetwas schiefgeht.

## Was die KI getan hat und was ich getan habe

Die Arbeitsteilung ist der Teil, den die Leute am schwersten glauben, also werde ich konkret. In zwölf Tagen habe ich etwa 110 Nachrichten geschrieben; die KI hat darauf mit rund 3.900 Tool-Aufrufen geantwortet -- Dateien lesen, Code schreiben, Tests ausführen, bauen, deployen. Sie hat fast den gesamten Code geschrieben, die Tests und die Beschreibungen der Pull-Requests.

Was sie nicht getan hat, ist entscheiden. Ich habe die Architektur und den Stack gewählt: Next.js 16 mit React 19, Prisma 7 auf PostgreSQL, TypeScript 7 mit seinem neuen nativen Compiler, SwiftUI für die Apple-Apps und eine Chrome-Erweiterung nach Manifest V3. Ich habe entschieden, dass die Isolation in der Datenbank durchgesetzt wird und nicht nur im Code, dass Passwörter an ihre Zeile gebunden sind, dass jede App nur ihren eigenen Teil bekommt und dass Tests jedes Release absichern. Ich habe auf echten Geräten getestet, die Regeln aufgeschrieben, denen die App folgen soll („Datum zuerst, Neuestes oben“, „keine schlichten Dropdowns, immer durchsuchbar“), und Nein gesagt -- oft. Als die iPad-Anmeldung aussah wie der Code-Bildschirm des Apple TV, war es ein Mensch, dem das auffiel.

> **Die Forschung sagt: Es kommt auf den Entwickler an**
> Ich will nicht so tun, als wäre die Beweislage einseitig. In einer randomisierten kontrollierten Studie von METR waren sechzehn erfahrene Open-Source-Entwickler mit KI-Werkzeugen vom Anfang des Jahres 2025 in ihren eigenen großen Codebasen um 19 % langsamer, während sie glaubten, schneller zu sein. METR nennt die Grenzen selbst -- eine kleine Stichprobe, eine Art von Arbeit, Werkzeuge, die sich seither verändert haben -- und hat inzwischen neuere Daten veröffentlicht. Die Lehre, die ich daraus ziehe, ist nicht, dass KI langsam oder schnell ist. Sondern dass der Gewinn von der Person abhängt, die sie steuert: davon, wie präzise sie spezifiziert, wie streng sie prüft und ob sie Tests und Reviews als Schranken behandelt. Ohne das bekommen Sie Vibes. Mit dem bekommen Sie Hearth.

## Was das beweist

Schnell und sorgfältig sind keine Gegensätze mehr. Dieselbe KI, die in zwölf Tagen 58.588 Zeilen geschrieben hat, hat auch die 313 Tests, die 34 Isolationsprüfungen, die 49 Trigger und das Sicherheits-Review geschrieben, die diese Zeilen ehrlich gehalten haben -- weil sie darum gebeten wurde und weil nichts in die Produktion

gelangte, ohne sie zu bestehen. Die Geschwindigkeit kam von den Werkzeugen. Die Sorgfalt kam aus dreißig Jahren Erfahrung damit, was in der Produktion schiefgeht, und aus der Weigerung, die langweiligen Teile auszulassen.

> KI hat das Engineering nicht abgeschafft. Sie hat die Ausrede abgeschafft, es auszulassen.

Wenn Sie diese Art von Geschwindigkeit für Ihr eigenes Produkt möchten -- für eine Delphi-Anwendung, die eine moderne Zukunft braucht, oder für ein System, das zu Ihrem Geschäft passt statt umgekehrt --, dann sehen Sie sich an, was ich gebaut habe und was ich anbiete, und dann sprechen wir darüber.

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 —**