NEWS · OCTOBER 19, 2026

Bouwen, niet kopen: maatwerksoftware in 12 dagen in plaats van nog een SaaS-licentie

Het huishouden van een makelaar draaide op spreadsheets, papier en een handvol abonnementen -- dus bouwde ik in twaalf dagen een eigen systeem voor haar, met AI die documenten archiveert, passkeys, een versleutelde kluis en apps voor iPhone, iPad, Apple TV, Mac en Chrome, en dit is wat dat betekent voor je Delphi-applicatie en je volgende keuze tussen kopen en bouwen.

By Dr. Holger Flick

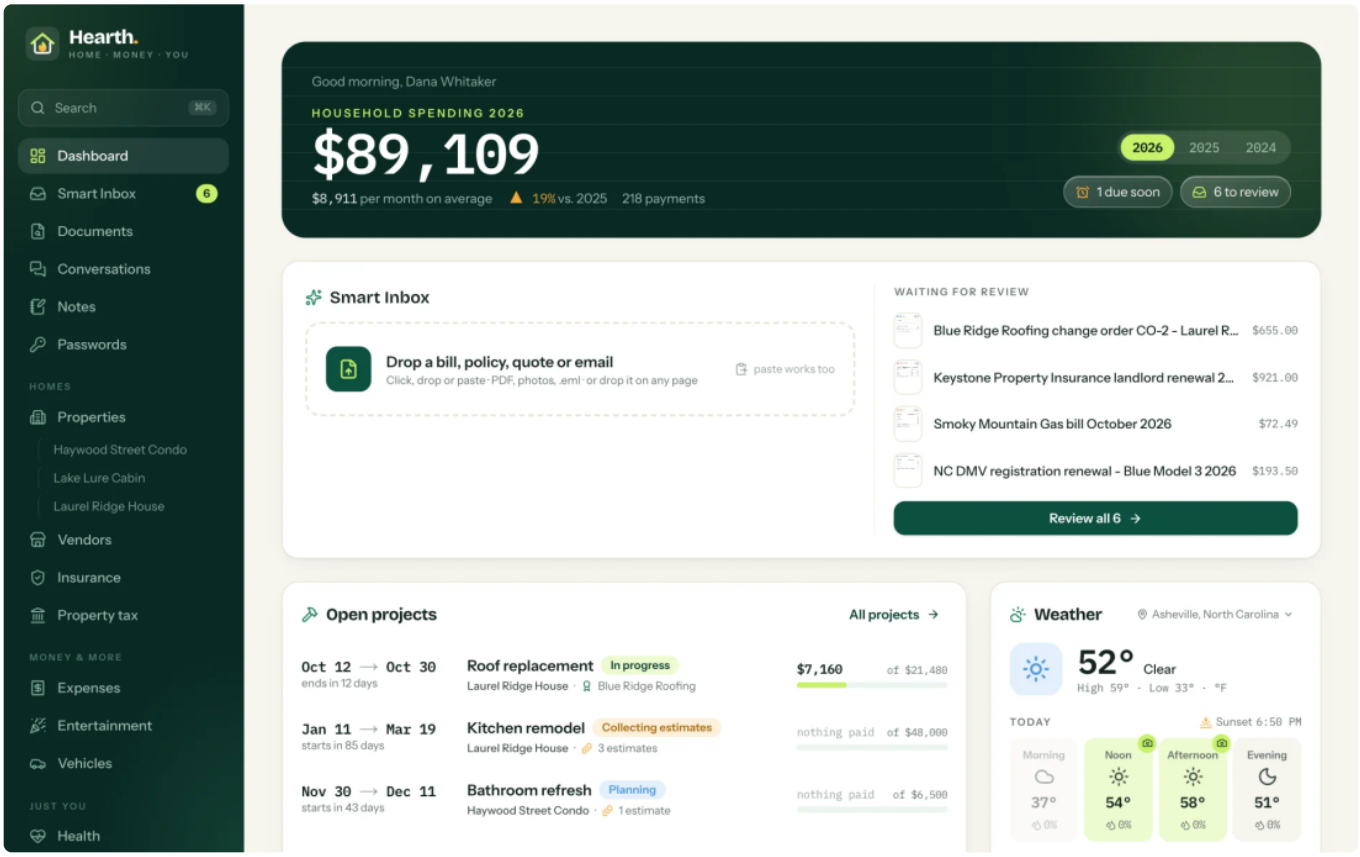
Mijn vrouw is makelaar. Jarenlang draaide ons huishouden zoals de meeste huishoudens -- en eerlijk gezegd de meeste kleine bedrijven -- draaien: een spreadsheet voor de verzekeringspolissen, een map met pdf's die niemand kon vinden, een agenda-app, een notitie-app, een wachtwoordmanager en een groeiende stapel papier op het aanrecht waar iedereen omheen liep. Elk van die tools deed zijn werk goed. Geen ervan wist iets van de andere, en geen ervan wist iets van *haar* werk.

Dus bouwde ik voor haar verjaardag één systeem in plaats daarvan. Het heet **Hearth**. Het leest elke rekening, polis en bon met AI, archiveert die bij de juiste woning, auto of het juiste project, beheert de agenda, de notities en de wachtwoorden, en laat alles zien op het web, op haar iPhone en iPad, op de Apple TV in de woonkamer, op de Mac en in Chrome. Van de eerste prompt tot de laatste release duurde het twaalf dagen.

Deze post is een rondleiding, en ook een pleidooi. Volgens mij is de vraag "welk product moeten we kopen?" tegenwoordig heel vaak de verkeerde eerste vraag. Voor steeds meer bedrijven is de betere vraag "hoe zou software eruitzien die precies bij ons past?" -- want de prijs van dat antwoord is ingestort. Ik laat je zien wat er gebouwd is, hoe lang het zonder AI had geduurd, waar mijn betoog standhoudt en waar niet, en wat dit allemaal betekent als je een Delphi-desktopapplicatie hebt die een tweede leven verdient.

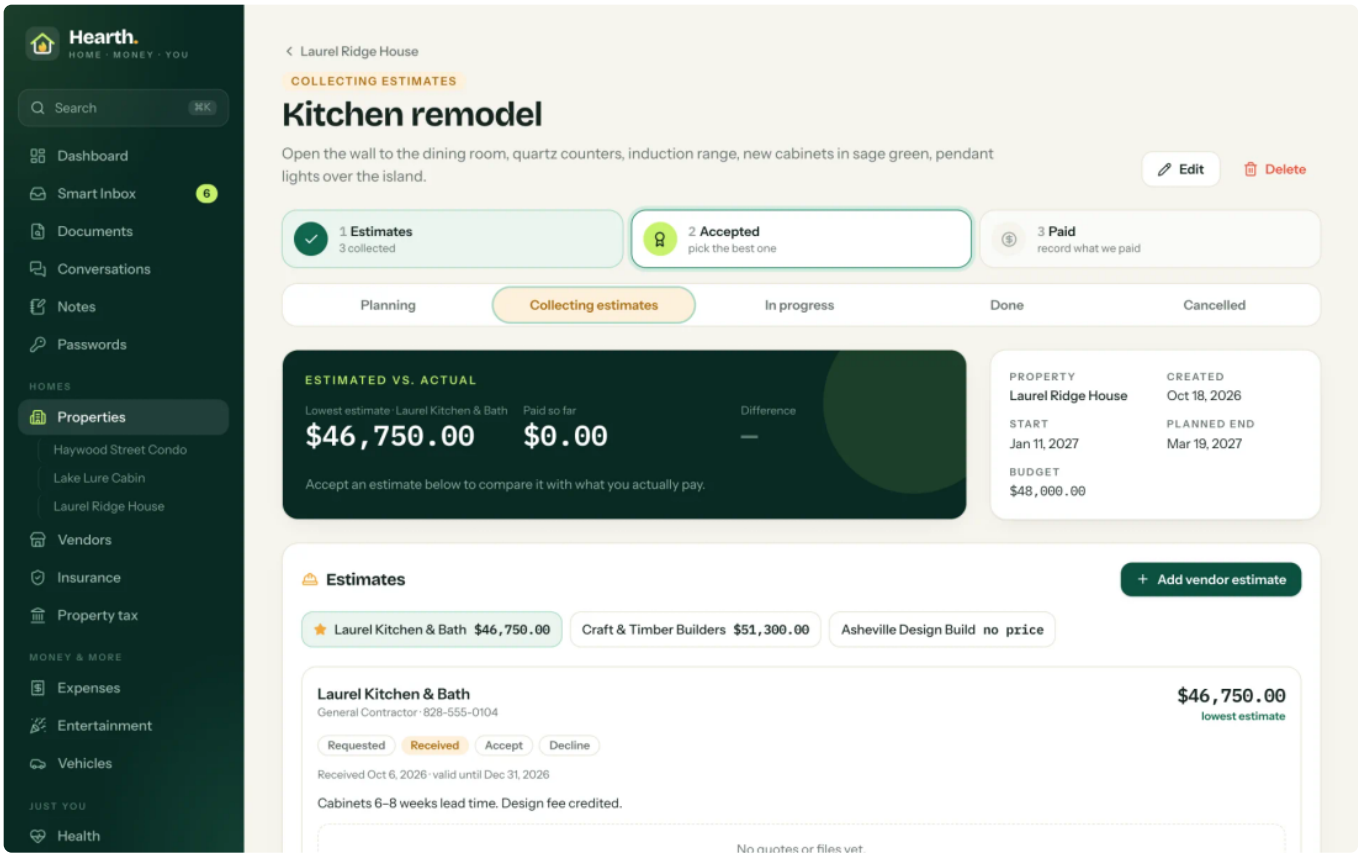
Wat Hearth is, in één scherm

Hearth beheert een huishouden met meerdere woningen en auto's zoals een klein bedrijf zijn bedrijfsvoering beheert. Elke dag begint op het dashboard: het weer (met een aantekening bij de dagen met goed licht voor woningfoto's), de lopende projecten, de uitgaven van het jaar en alles wat op een beslissing wacht.



Het dashboard van een fictief demohuishouden: het weer met de dagen met goed licht voor foto's, lopende projecten en de uitgaven van het jaar.

Achter het dashboard zitten de onderdelen die vroeger losse tools of losse mappen waren. Woningen komen met hun HOA (zeg maar de Amerikaanse VvE), de aanslagen onroerendezaakbelasting met hun termijnen, en de projecten met elk meerdere offertes van aannemers. Auto's komen met hun leasecontracten, verzekeringen met hun looptijden, betalingsregelingen en een tijdlijn van elke polis, en de zorgverzekering met haar rekeningen. Leveranciers hebben hun contactpersonen en telefoonnummers, gesprekken verzamelen e-mails en opgenomen telefoontjes, en terugkerende uitgaven boeken zichzelf. Notities, wachtwoorden en een volwaardige agenda maken het plaatje compleet.

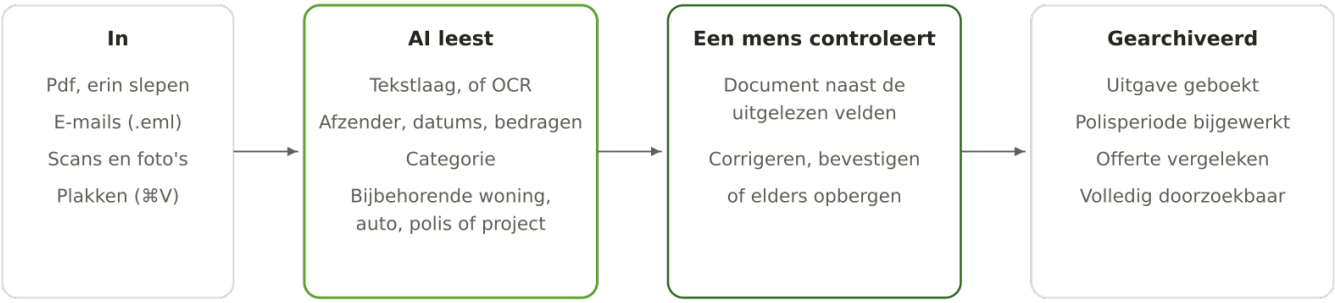


Een project verzamelt offertes: drie aannemers naast elkaar, de laagste gemarkeerd (demodata).

Niets hiervan is exotisch. Dat is precies het punt. Elke afzonderlijke functie bestaat ergens in een commercieel product. Wat niet bestaat, is dat ene product met precies deze functies, precies zo met elkaar verbonden, voor precies deze persoon.

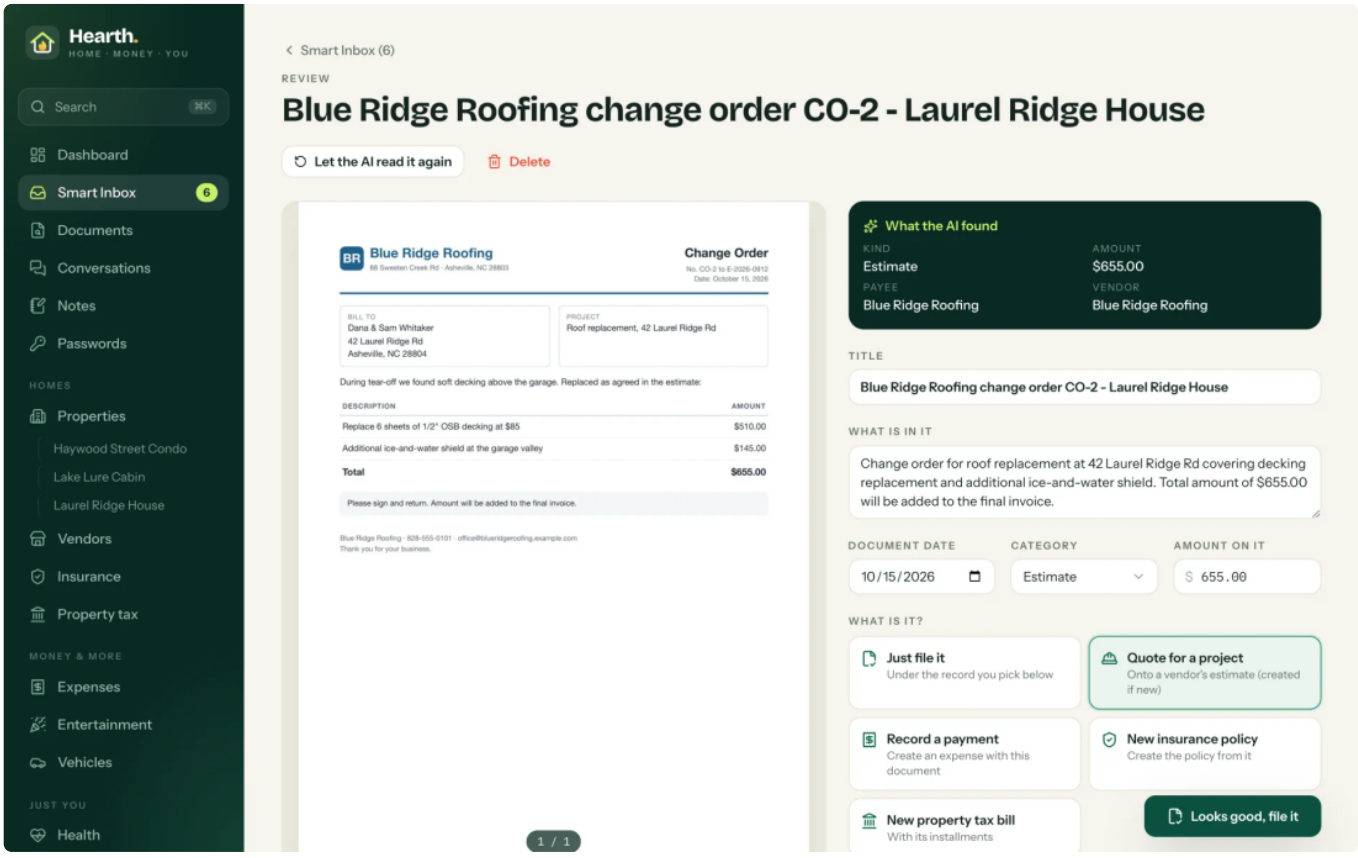
De stapel papier, opgelost: OCR, categorieën en archiveren

De functie die het aanrecht veranderde, is de documentenpijplijn. Een document komt binnen als pdf, foto, scan of opgeslagen e-mail met bijlagen, op de pagina gesleept of geplakt met V. Als een pdf zijn eigen tekst bevat, leest Hearth die; is het een scan of een foto, dan gaat elke pagina als OCR door een vision-model. Daarna haalt de AI eruit wat ertoe doet: wie het stuurde, de datum, het bedrag, de vervaldatum, het polis- of rekeningnummer. Vervolgens stelt hij voor waar het document thuishoort: welke categorie, welke woning of auto, welke polis of welk project.



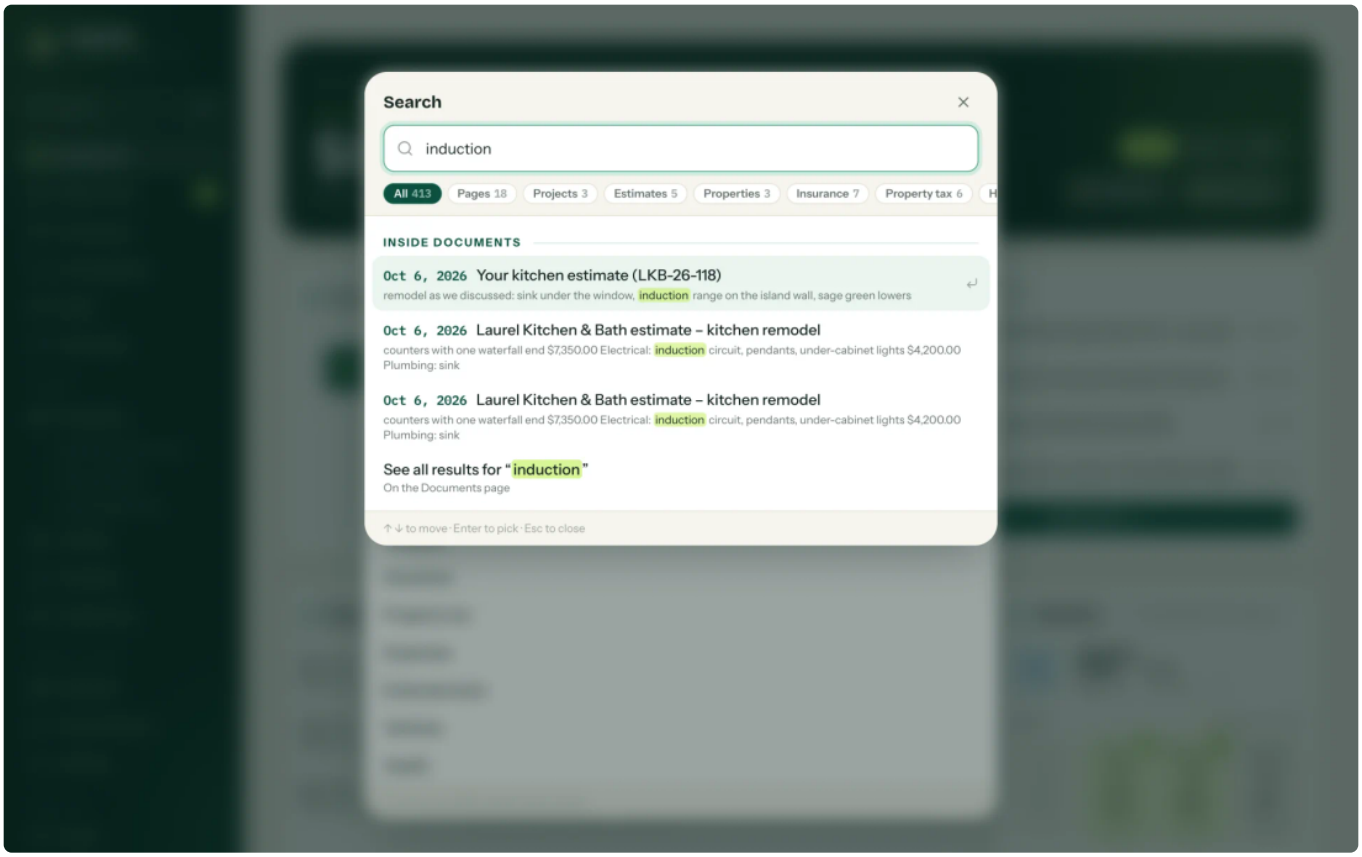
Van stapel papier naar een gearchiveerd, doorzoekbaar document -- met een mens die meebeslist

Het derde vak is het vak waar het mij het meest om gaat. De AI beslist nooit zelf waar iets heen gaat: alles wat in de Smart Inbox binnenkomt, wacht tot een mens het voorstel bevestigt of corrigeert. (Sleep je een document rechtstreeks op een project of een polis, dan heb je al gezegd waar het hoort, en slaat het de wachtrij over.) Het controlescherm toont het document direct naast wat de AI eruit heeft gelezen, dus nakijken kost seconden in plaats van minuten.



De Smart Inbox: aan de ene kant het document, aan de andere kant wat de AI las en waar hij voorstelt het op te bergen.

Zodra een document gearchiveerd is, gaat het echt aan het werk. Een rekening wordt een uitgave bij de juiste woning, een verlenging van een verzekering verlengt de juiste polis, en de offerte van een aannemer belandt naast de andere offertes voor hetzelfde project, zodat de goedkoopste actuele aanbieding meteen opvalt. Elk woord in elk document is overal doorzoekbaar met K, en dat is meestal het moment waarop mensen ophouden met vragen waarom ze dit zouden willen.

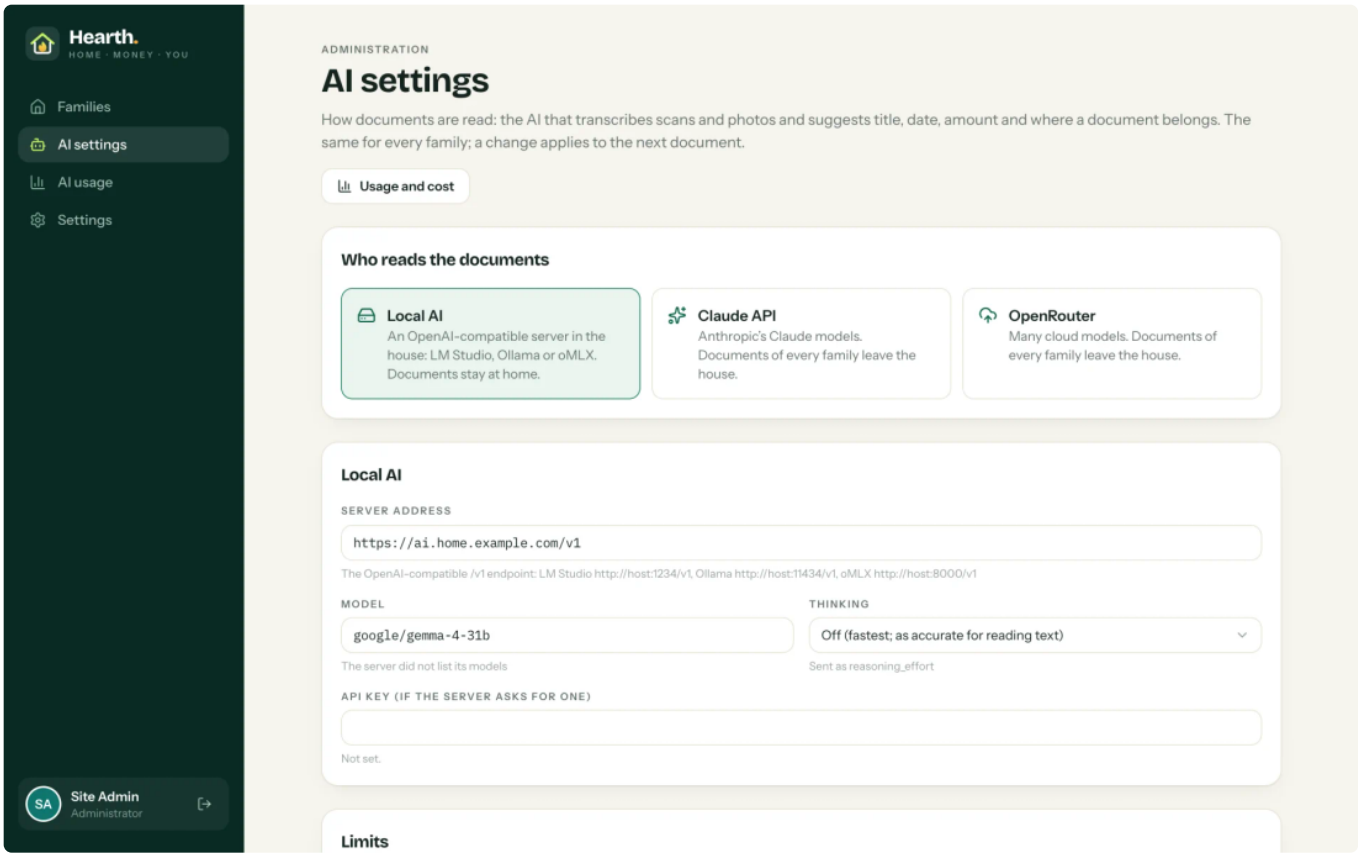


K doorzoekt elk woord in elk document, waar het ook is opgeborgen.

Telefoongesprekken krijgen dezelfde behandeling. Tik na een gesprek met de dakdekker op de microfoon, vertel waar het over ging, en Hearth zet het om in tekst als gesprek met die leverancier. De transcriptie draait op een Whisper-server op een Mac in huis, dus de opname verlaat het huis nooit.

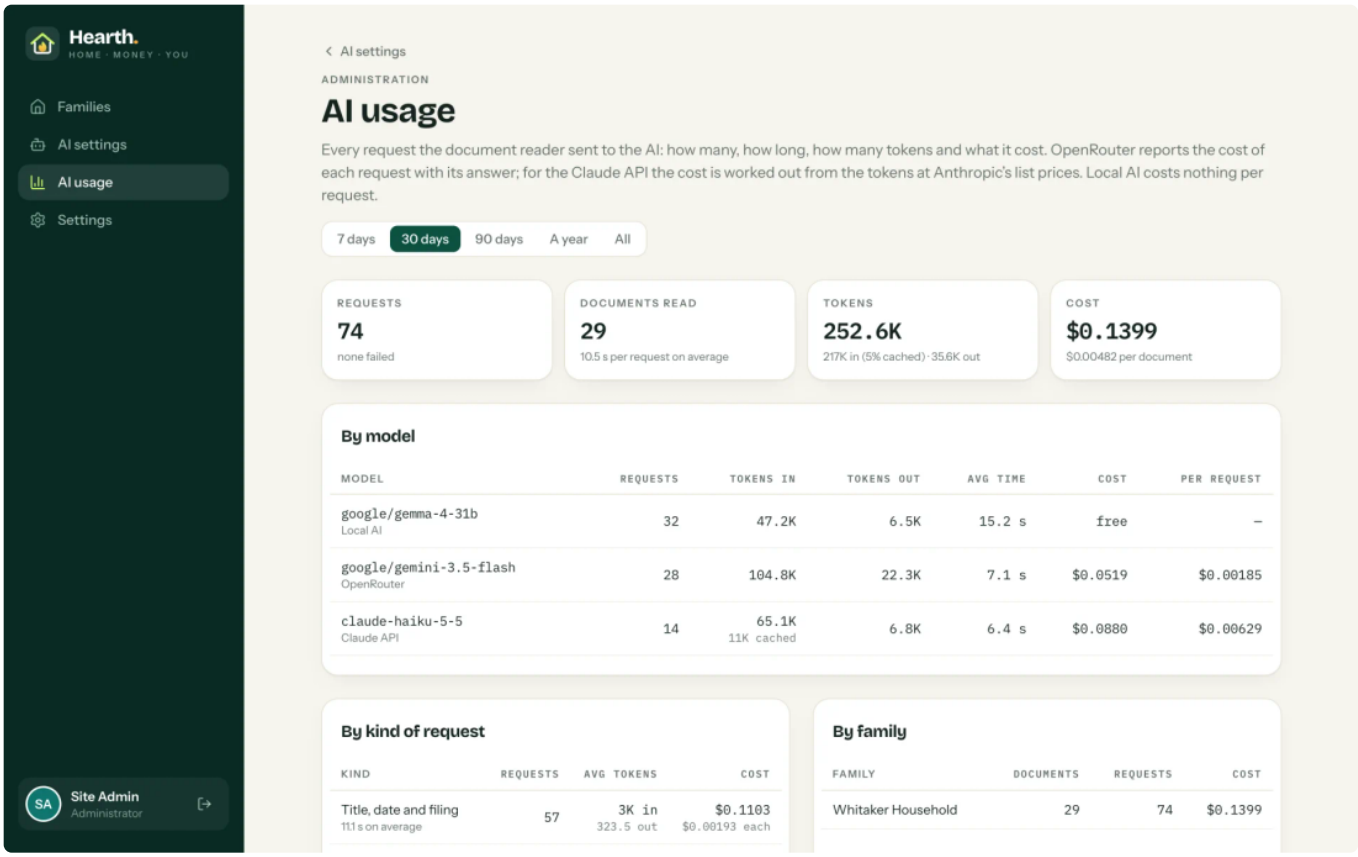
Elke AI-provider, ook een die in huis woont

Het huishouden kiest zelf, in de beheerinstellingen en zonder nieuwe release, hoe documenten worden gelezen. Hearth praat met drie soorten providers: de Claude API van Anthropic, elk model op OpenRouter (dat de modellen van OpenAI, Google, Anthropic, Mistral en vele anderen achter één API zet), of een lokaal model op een willekeurige OpenAI-compatibele server zoals LM Studio of Ollama. Met de lokale optie verlaat geen enkele pagina het huis -- daar schreef ik voor Delphi uitgebreid over in Give Your Delphi App a Brain.



De AI-instellingen: Claude, elk model op OpenRouter, of een lokaal model dat elke pagina in huis houdt.

Omdat "AI" zonder rekening erbij een sprookje is, wordt elk verzoek gelogd met zijn tokens, zijn duur, zijn fouten en zijn kosten. Cloudmodellen melden of schatten hun prijs; het lokale model kost niets behalve stroom. De beheerder ziet het per model, per soort verzoek, per gezin en per dag.

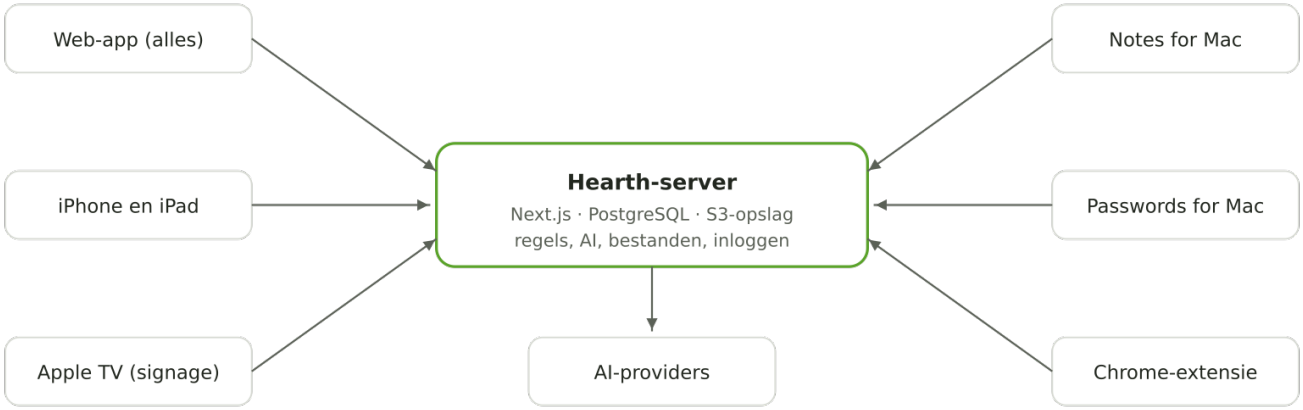


AI-gebruik en kosten: verzoeken, tokens en kosten per model en aanbieder -- het lokale model kost niets (demodata).

> **Waarom vrije keuze van provider ertoe doet voor je product**
> Als je vandaag AI in een product bouwt, komt het beste model van volgend jaar zeer waarschijnlijk van een andere leverancier dan dat van dit jaar. Een applicatie die de provider als instelling behandelt en niet als architectuurbeslissing, kan in één middag overstappen. Het stelt bovendien een klant met strenge privacyregels in staat alles lokaal te draaien, en dat is op zichzelf al een verkoopargument.

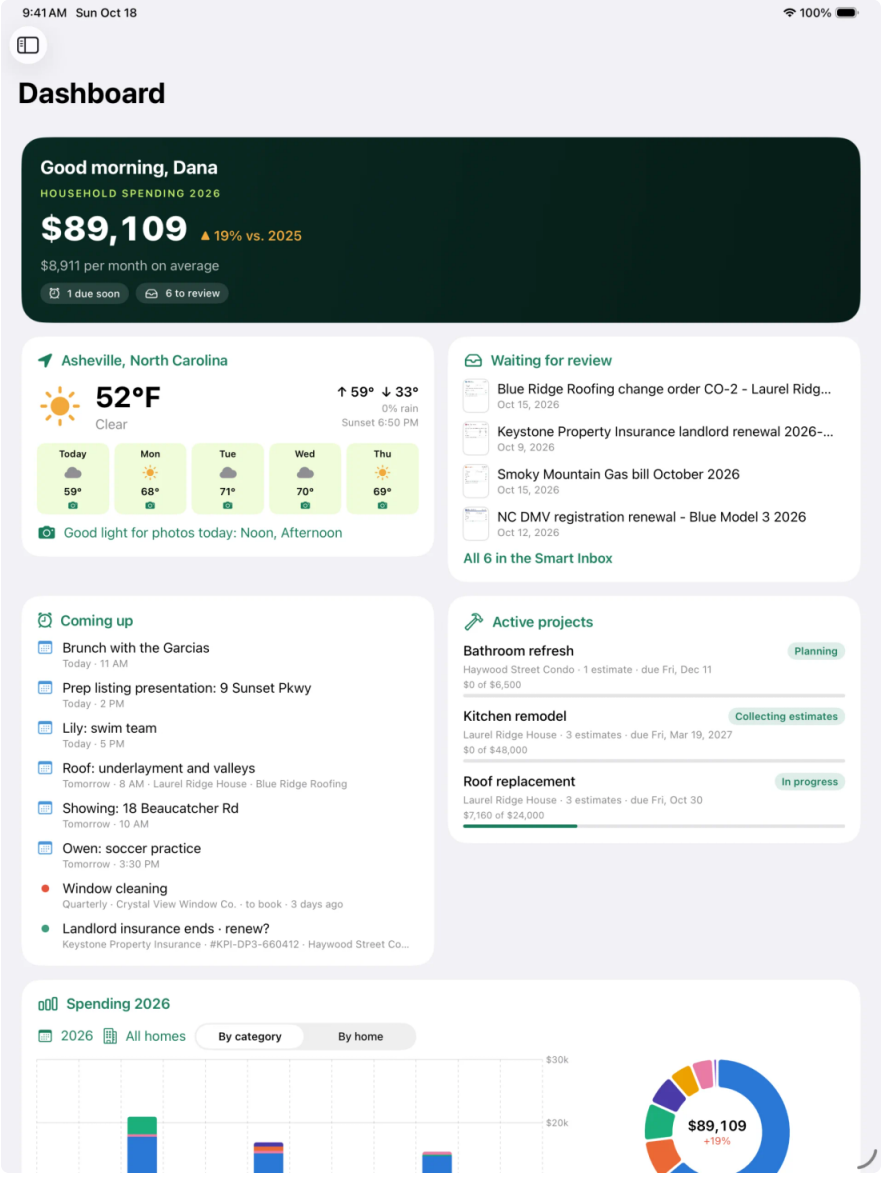
Eén backend, zeven voordeuren

Hearth is niet één app; het is één server met meerdere ingangen. De webapplicatie kan alles. Een app voor iPhone en iPad, die alleen leest, toont elk onderdeel ervan. Een app voor Apple TV maakt van het scherm in de woonkamer een gezinsdashboard. Notes for Mac en Passwords for Mac zijn volwaardige native vervangers van de apps die ze vervingen, en een Chrome-extensie vult inloggegevens in op de websites zelf.



Eén server, één database, één set regels -- en zeven ingangen

Het diagram ziet er eenvoudig uit omdat het moeilijke deel onzichtbaar is: elke voordeur gaat op de server door dezelfde regels. Elke app logt zelfstandig in, met een code die op het web wordt bevestigd, en elke soort app mag alleen doen waarvoor hij gemaakt is. De Chrome-extensie kan wachtwoorden lezen en opslaan, en verder niets; als iemand zijn inlogsessie uit de browser zou stelen, kon hij nog geen enkel document openen.

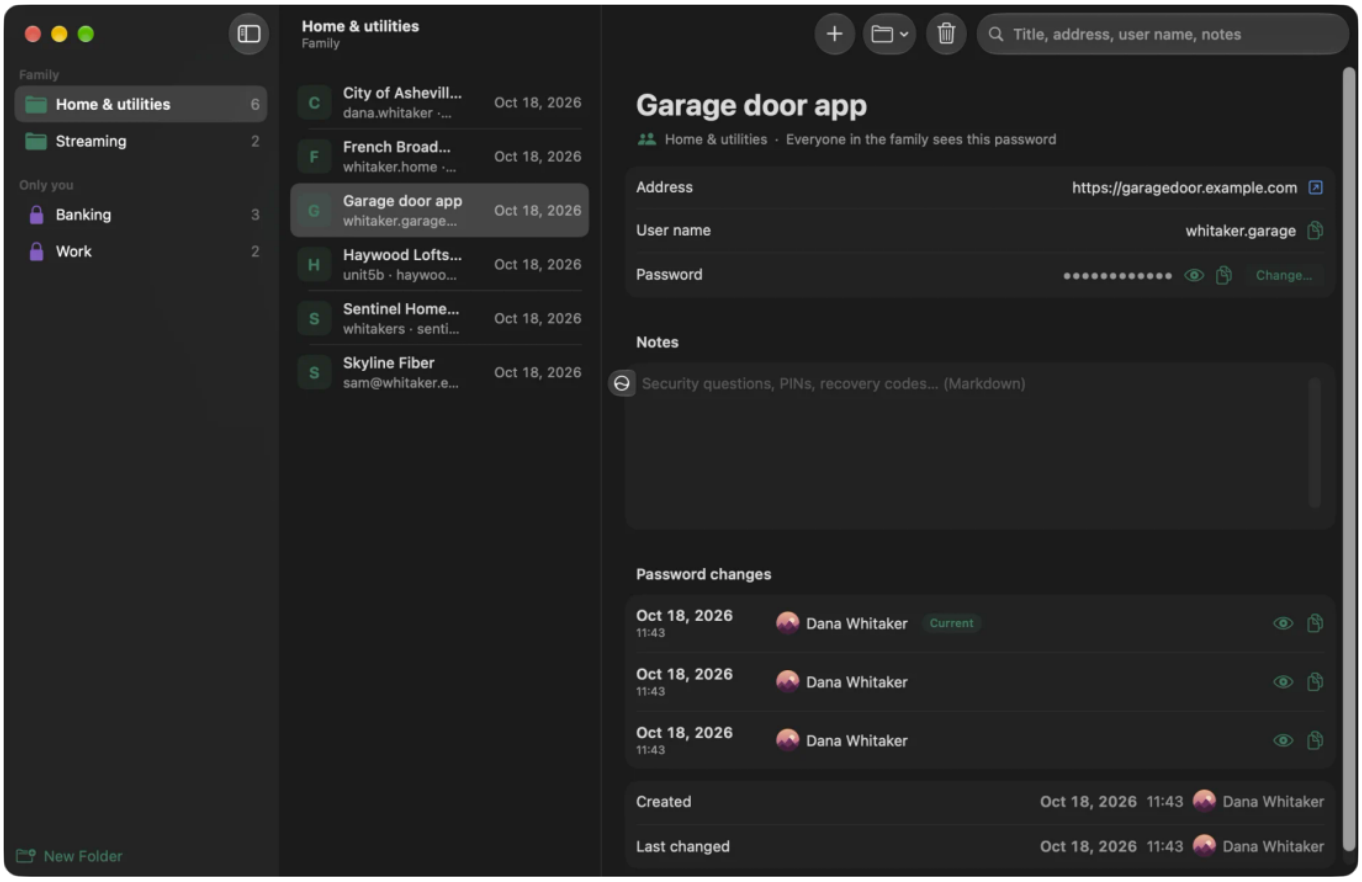


De iPad-app toont elk onderdeel van Hearth, alleen-lezen, met documenten in Quick Look.

De Apple TV-app verdient een apart woord, want daar moeten mensen om glimlachen. Hij toont de afspraken van vandaag, de maand, het weer met de uren met goed licht voor foto's en de uitgaven van het jaar, en draait als diavoorstelling die de schermbeveiliging op afstand houdt. Thuis is dat een gezinsdashboard. In een kantoor, een showroom of een wachtkamer heet precies hetzelfde idee **digital signage**, live gevoed met je eigen data in plaats van met een usb-stick die iemand eens per kwartaal bijwerkt.

De Apple TV-app: vandaag, de agenda, het weer en de uitgaven -- een gezinsdashboard, of digital signage op kantoor.

Op de Mac vervingen de twee native apps de notitie- en wachtwoordapps die we eerder gebruikten. De wachtwoorden blijven verborgen achter Touch ID en vergrendelen na vijf minuten weer, en de Chrome-extensie biedt aan een login op te slaan of bij te werken zodra je ergens inlogt.

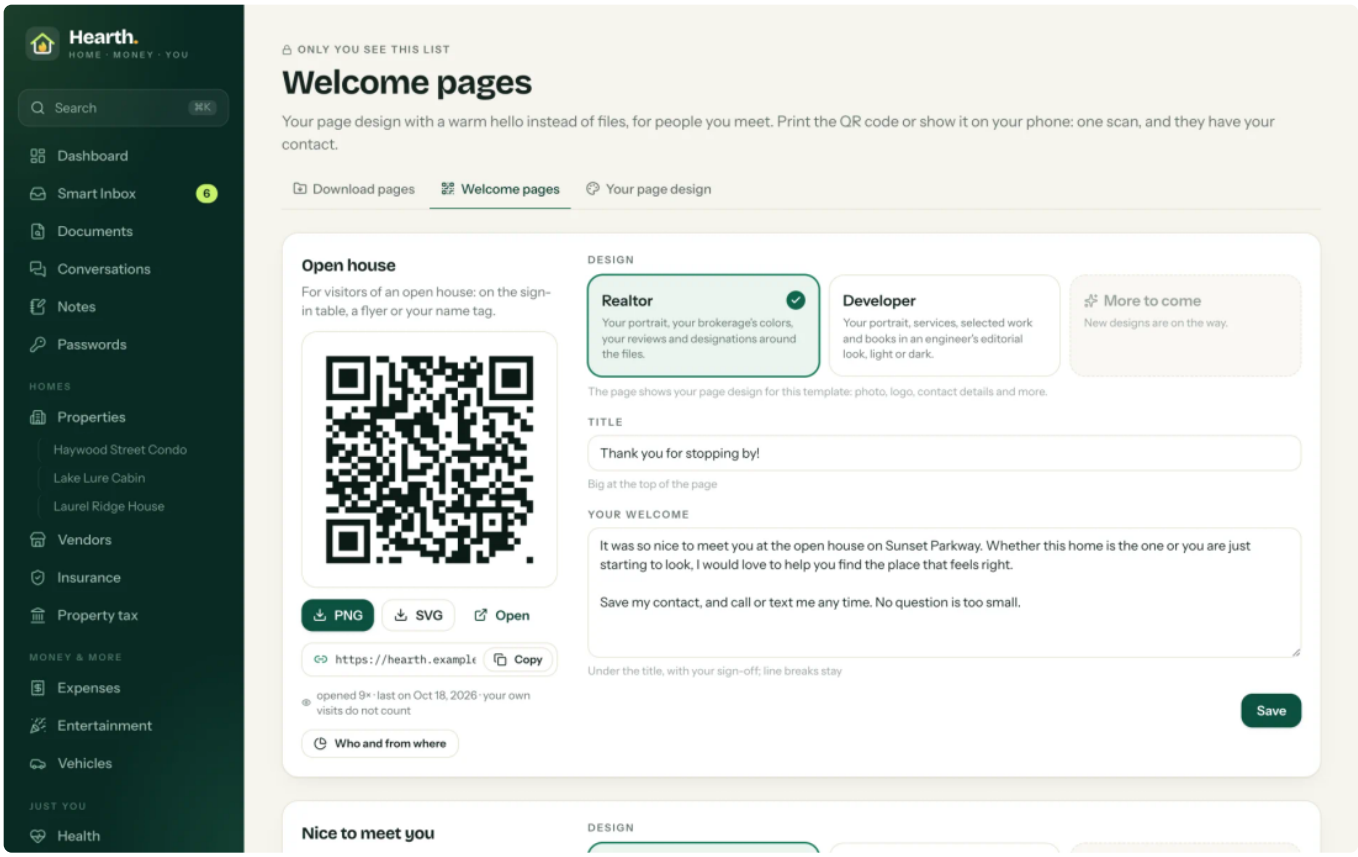


Passwords for Mac: vergrendeld achter Touch ID, met wachtwoordgeschiedenis en een generator.

Gebouwd voor één makelaar, niet voor een miljoen gebruikers

Hier bewijst maatwerksoftware zijn waarde. Een generiek product moet iedereen bedienen, dus bedient het niemand precies. Hearth weet dat zijn eigenaar huizen verkoopt, en heeft daarom dingen die geen enkele huishoud-app ooit zou uitbrengen.

Welkomstpagina's zijn een goed voorbeeld. Een makelaar kan een pagina maken voor een klant of een open huis -- vanuit een sjabloon, met een QR-code om te printen -- en Hearth logt wanneer die werd geopend, ongeveer vanaf waar en met welke browser. De weerkaart markeert de dagen met goed licht voor woningfoto's. Niets hiervan kostte veel tijd om te bouwen. Elk ervan zou een feature request zijn in de backlog van een groot product, waarover gestemd wordt door duizenden gebruikers die het niet nodig hebben.



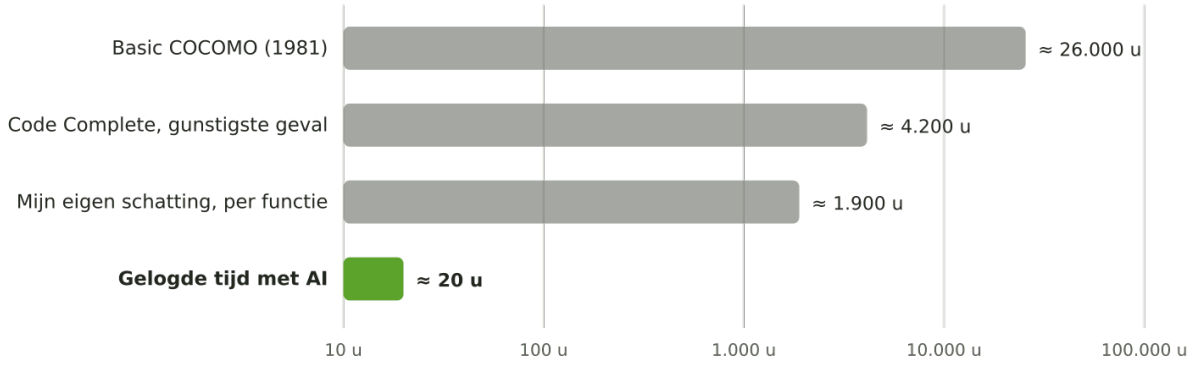
Een welkomstpagina met QR-code, in een minuut gemaakt vanuit een sjabloon (demodata).

> **Hetzelfde patroon, in een bedrijf**
> Vervang "makelaar" door "verzekeringsadviseur", "vastgoedbeheerder", "tandtechnisch lab" of "machinebouwer", en het verhaal blijft hetzelfde. Een bedrijf dat een groot CRM koopt, betaalt voor honderden functies die het nooit gebruikt en betaalt daarna nog een keer -- aan consultants of aan omwegen -- voor de vijf die het echt nodig heeft. Een systeem op maat begint bij die vijf.

Hoe lang had dit zonder AI geduurd?

Twaalf dagen is de kalendertijd van de eerste prompt op 7 oktober tot de laatste release op 18 oktober. Mijn Claude Code-sessielogs laten in die periode ongeveer **20 uur** actieve werktijd met de AI zien. Dat getal telt alleen de tijd waarin een sessie bezig was, zonder pauzes van meer dan twintig minuten, dus testen op de iPad of nadenken op de veranda zit er niet in.

Het resultaat is **58.588 niet-lege regels** TypeScript, Swift, SQL en JavaScript, 52 gemergede pull requests en 48 getagde releases. Regels code zijn een slechte maatstaf voor waarde -- Edsger Dijkstra betoogde beroemd genoeg dat je ze moet boeken als "verbruikte regels" en niet als "geproduceerde regels" -- maar ze zijn de enige maat voor omvang die de klassieke schattingsmodellen accepteren, dus laten we het ze vragen.



Geschatte inspanning om Hearth met de hand te bouwen, tegenover de gelogde tijd met AI (uren, logaritmische schaal)

De grafiek heeft een logaritmische schaal nodig, en dat zegt op zichzelf al iets. Zo komt elke balk tot stand.

1. **Basic COCOMO.** Barry Boehms Constructive Cost Model uit 1981 schat de inspanning voor een "organic" project op 2,4 × KLOC^1,05 persoonsmaanden. Voor 58,6 duizend regels is dat ongeveer 172 persoons-maanden, oftewel zo'n 26.000 uur bij de 152 uur per maand van het model. Het model is oud en gaat uit van een wereld zonder moderne frameworks, dus ik zie het als het plafond.
2. **Code Complete, gunstigste geval.** Steve McConnells *Code Complete* (2e editie) noemt 2.000 tot 25.000 regels per manjaar voor projecten van 10.000 regels, en 1.000 tot 20.000 voor 100.000 regels (tabel hier overgenomen). Neem je de meest optimistische kant, 25.000 regels per jaar, dan is Hearth ongeveer 2,3 manjaar, zo'n 4.200 uur bij 1.800 werkuren per jaar.
3. **Mijn eigen schatting.** Ik heb het geschat zoals ik klantprojecten schat: functie voor functie, voor één ervaren ontwikkelaar, met moderne frameworks en zonder AI. Het datamodel en inloggen met passkeys en gezin-sisolatie: 25 dagen. De documentenpijplijn met de AI-providers: 25. De huishoudmodules, van woningen tot zorgverzekering: 55. Agenda, dashboard, zoeken, notities, wachtwoordkluis en welkomstpagina's: 44. Tests, de securityreview en de deployment: 17. De app-API, de app voor iPhone en iPad, de Apple TV-app, beide Mac-apps en de Chrome-extensie: 69. Dat is 235 werkdagen, of iets minder dan 1.900 uur -- en het is bewust de meest optimistische van de drie.

Zelfs tegenover mijn eigen optimistische schatting is het verschil ongeveer twee ordes van grootte. Ik weet hoe dat klinkt, dus in het volgende stuk spreek ik mezelf tegen.

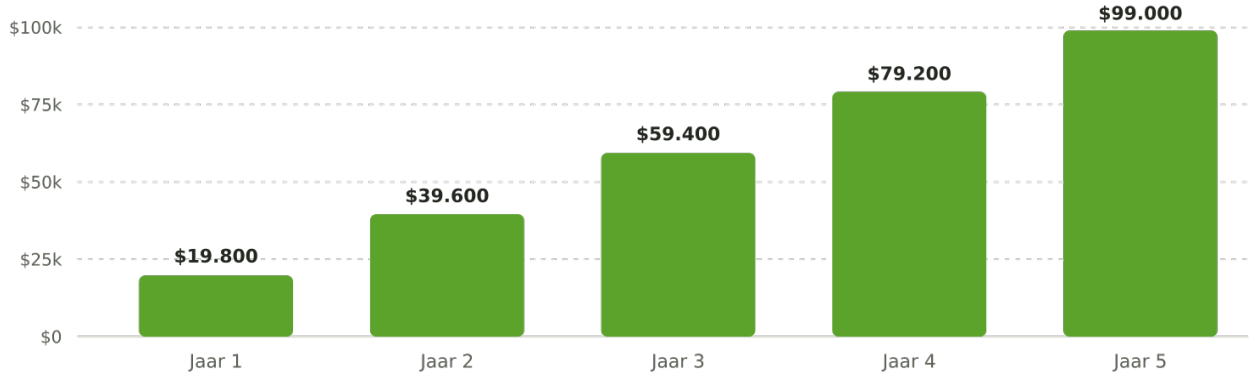
Waar deze vergelijking niet opgaat

De 20 uur zijn sessietijd, geen volledige verantwoording van mijn dag: nadenken, testen op apparaten en beslissen wat er gebouwd moest worden gebeurde eromheen. De AI schreef veel meer code dan ik zelf zou hebben getypt, en meer regels is niet meer waarde. Een greenfieldproject met één veeleisende gebruiker en geen legacydata is het gunstigste geval voor AI -- een migratie met twintig jaar aan bedrijfsregels verstopt in een Delphi 7-codebase is een heel ander beest, zoals ik beschreef in Adapt or Disappear. En het onderzoek is niet eenduidig: in een gerandomiseerde studie van METR waren ervaren opensource-ontwikkelaars met AI-tools van begin 2025 19% *langzamer* in hun eigen volwassen codebases (METR merkt zelf op dat die resultaten achterhaald zijn), en het DORA-rapport van 2024 vond dat de inzet van AI samenging met een lagere stabiliteit van opleveringen. Mijn conclusie is niet "AI maakt iedereen 100 keer sneller". Ze is beperkter: een ervaren ontwikkelaar die precies specificeert, alles reviewt en op tests staat, kan nu in dagen bouwen waar vroeger een jaar voor nodig was.

Hoe ik voorkom dat het "vibe coding" wordt -- de tests, de securityreview, de releasepoort, de isolatiecontroles -- is het onderwerp van mijn volgende post, Geen vibe coding.

Kopen of bouwen? Het eerlijke antwoord is "dat hangt ervan af"

Er is een eenvoudige rekensom die veel bedrijven nooit maken. Een CRM met een grote naam vraagt voor zijn enterprise-editie volgens de prijslijst zo'n **$165 tot $175 per gebruiker per maand**, afhankelijk van welke prijstracker je het vraagt (Capterra, Redress Compliance). Voor tien gebruikers is dat ongeveer $20.000 per jaar aan alleen licenties, nog vóór add-ons, integraties en de consultant die het inricht.



Cumulatieve licentiekosten volgens prijslijst voor tien CRM-gebruikers à $165 per gebruiker per maand (exclusief add-ons en implementatie)

Ruwweg $100.000 in vijf jaar, voor software die dan nog steeds niet weet wat je bedrijf eigenlijk doet. Naar mijn idee koop je voor dat budget nu een systeem op maat, met geld over -- een systeem dat van jou is, dat jouw vakjargon spreekt en dat meegroeit met je bedrijf in plaats van met de roadmap van de leverancier.

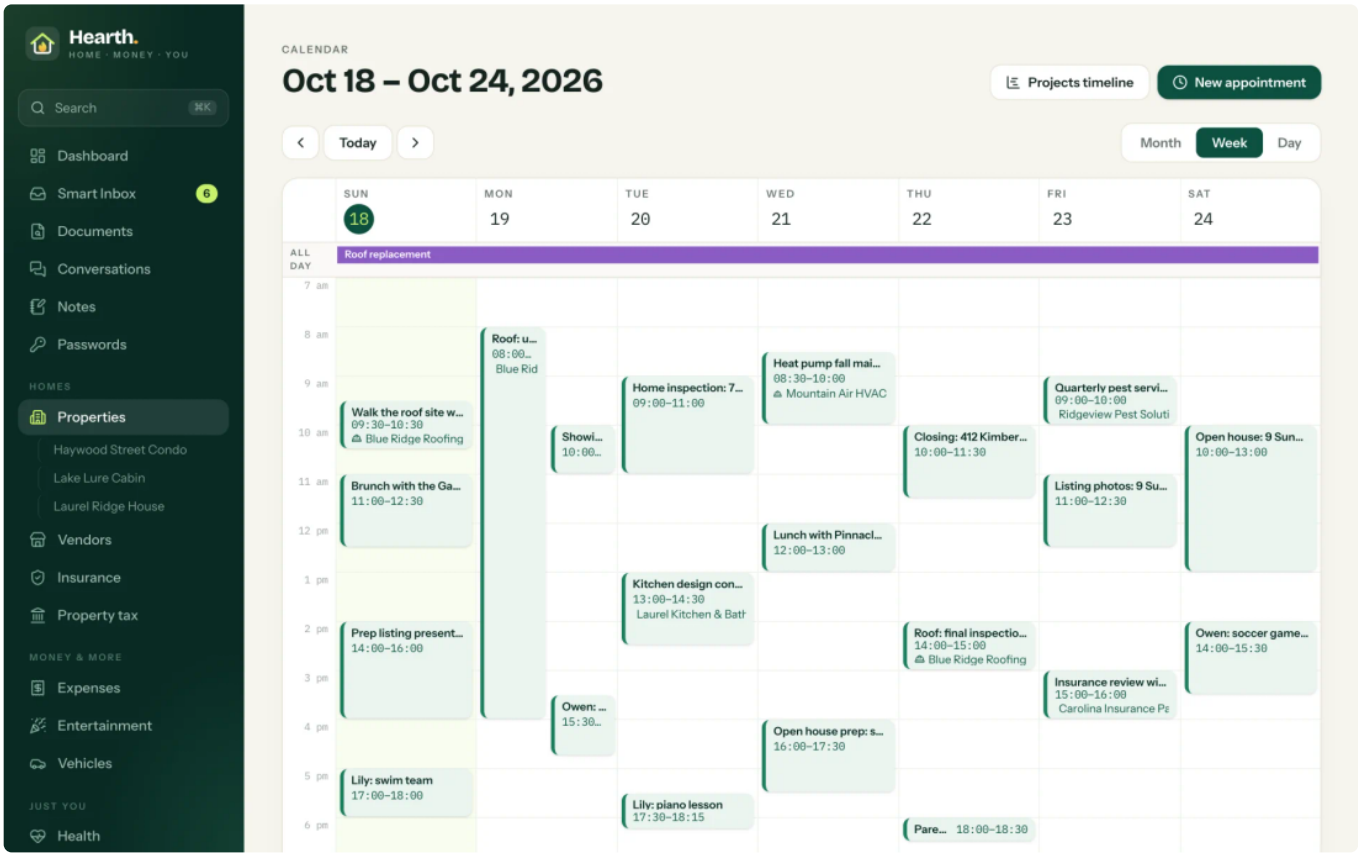
Dat gezegd hebbende: een abonnement levert echte dingen op die je makkelijk vergeet. Hosting, updates, compliancewerk, support om twee uur 's nachts en integraties met duizend andere producten. Maatwerksoftware vraagt onderhoud, back-ups en iemand die opneemt als er iets stukgaat. Als je proces standaard is en een product erbij past, koop dan dat product -- ik meen het. En spreekt het idee je aan om je eigen systeem te bezitten, maar wil je niet bij nul beginnen, dan zijn er uitstekende opensource-CRM's zoals Odoo Community, SuiteCRM, EspoCRM en Twenty, die je kunt aanpassen en hosten waar je maar wilt.

> **Wanneer je moet bouwen**
>
> Bouw als het proces *zelf* je voorsprong is: als je mensen hun dag besteden aan het omzeilen van een tool, als de data op vijf plekken staat, of als die ene functie die je nodig hebt precies de functie is die geen leverancier ooit prioriteit zal geven. Koop als het proces handelswaar is -- salarisadministratie, e-mail, boekhouding -- en het product het al goed doet.

Wat dit betekent voor je Delphi-applicatie

De meeste van mijn klanten komen niet bij me met een leeg vel papier. Ze komen met een Delphi-applicatie die al vijftien of vijfentwintig jaar hun bedrijf draaiende houdt. Die werkt, de gebruikers zijn er dol op, en langzaam wordt ze de reden dat het bedrijf niets nieuws kan: geen webtoegang, geen telefoon, geen tablet, geen AI, en een groep ontwikkelaars die niet bepaald groter wordt.

Hearth laat zien wat zo'n applicatie kan worden, en er zijn meerdere eerlijke wegen naartoe. Je kunt de Delphi-code houden en er een REST API voor zetten -- met TMS XData of met niets meer dan WebBroker -- en daar vervolgens frontends voor web, mobiel en tv tegenaan bouwen. Je kunt de webpagina's vanuit Delphi zelf renderen met WebStencils. Of je verhuist het geheel naar een moderne webstack, de weg die ik beschreef in mijn serie Delphi naar Next.js. Het juiste antwoord hangt af van je code, je team en je gebruikers, en ik vertel je liever welke weg past dan dat ik je de weg verkoop die ik zelf het leukst vind.



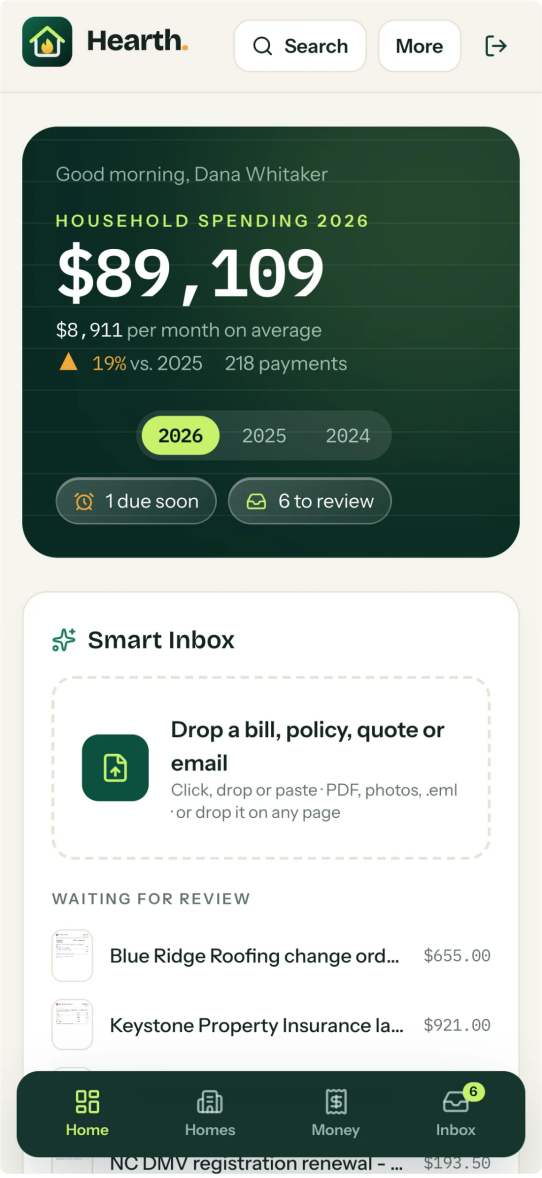
De agenda met afspraken die je kunt verslepen, van grootte kunt veranderen en met één klik kunt toevoegen.

Voor het moderniseren van een website geldt dezelfde regel. Je merk is geen sjabloon, dus ik laat je kleuren, je typografie en je logo staan en moderniseer wat eronder zit: snelheid, telefoons, toegankelijkheid, vindbaarheid en een contentworkflow die niet voor elke komma een ontwikkelaar nodig heeft. Heb je al een designsysteem, dan werken we daarbinnen.

Teams en solo-ontwikkelaars

Ik werk op beide manieren, en beide komen veel voor. Sommige klanten zijn een solo-ontwikkelaar die al jaren een product draagt en een partner zoekt die dit eerder heeft gedaan -- iemand die het plan reviewt, de delen overneemt die niet bij zijn vaardigheden passen, of samen de eerste module bouwt. Andere zijn teams die de moderne stack en een werkend AI-ondersteund proces in hun eigen repository willen invoeren, waarbij hun eigen mensen het werk doen en ik meekijk en review. In beide gevallen gebeurt het werk in een repository die je vanaf dag één kunt inzien, in kleine releases die je zelf kunt draaien.

We beginnen altijd op dezelfde manier en, zoals op mijn pagina met diensten staat, kost dat niets: een gesprek van dertig tot zestig minuten over het systeem en waar het knelt. Voor bestaande code is de volgende stap een assessment met een vaste scope -- ik lees de code, compileer hem, draai hem, en jij krijgt een schriftelijk plan en een realistische schatting die van jou zijn, of we nu verdergaan of niet. Meer systemen die ik heb opgeleverd vind je onder Work.



De web-app op een telefoon: hetzelfde systeem, ingedeeld voor het kleine scherm.

De conclusie

Hearth begon als verjaardagscadeau en werd het beste argument voor maatwerksoftware dat ik kan geven. In twaalf dagen bouwde één ervaren ontwikkelaar met AI wat de handboeken ergens tussen één en meerdere persoonsjaren schatten: AI die documenten leest en archiveert met elke provider, passkeys en een versleutelde kluis, en native apps voor iPhone, iPad, Apple TV en de Mac, plus Chrome. De AI nam de beslissingen niet; hij maakte het goedkoop om ze uit te voeren.

> De vraag is niet langer of je maatwerksoftware kunt betalen. De vraag is of je het je kunt veroorloven je bedrijf te blijven buigen naar de software van een ander.

Heb je een Delphi-applicatie die een toekomst verdient, of een proces waar geen enkel product bij past, dan hoor ik er graag over. Laten we praten.

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 —