NEWS · OCTOBER 19, 2026

Développer plutôt qu'acheter : un logiciel sur mesure en 12 jours au lieu d'une licence SaaS de plus

Le foyer d'une agente immobilière tournait sur des tableurs, du papier et une poignée d'abonnements -- alors je lui ai construit son propre système en douze jours, avec classement des documents par IA, passkeys, coffre-fort chiffré et applis pour iPhone, iPad, Apple TV, Mac et Chrome, et voici ce que cela signifie pour votre application Delphi et votre prochaine décision « acheter ou développer ».

By Dr. Holger Flick

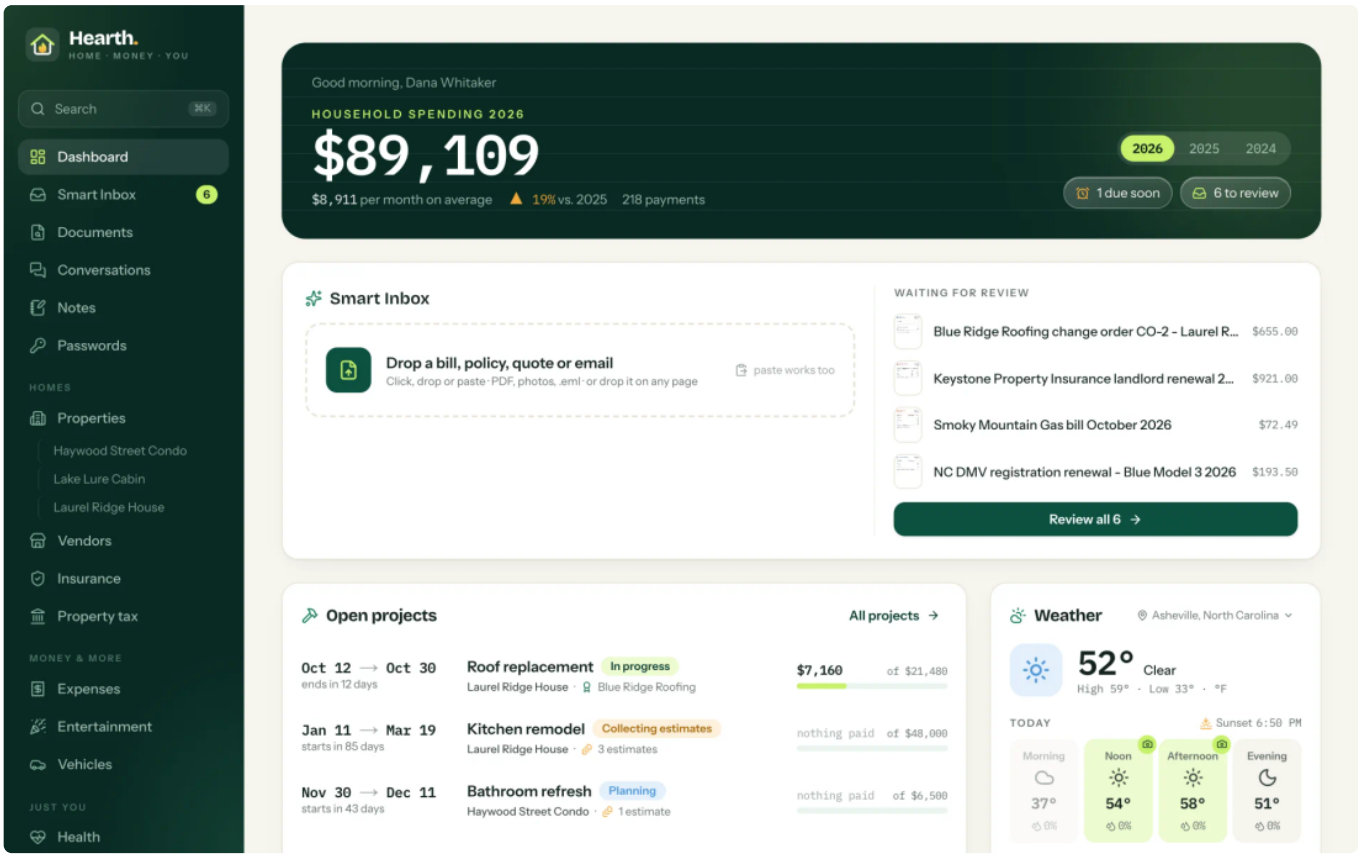
Ma femme est agente immobilière. Pendant des années, notre foyer a fonctionné comme la plupart des foyers -- et, pour être franc, comme la plupart des petites entreprises : un tableur pour les contrats d'assurance, un dossier de PDF que personne ne retrouvait, une appli de calendrier, une appli de notes, un gestionnaire de mots de passe, et une pile de papier toujours plus haute sur le comptoir de la cuisine, que tout le monde contournait. Chacun de ces outils faisait bien son travail. Aucun ne connaissait les autres, et aucun ne savait quoi que ce soit de *son* métier à elle.

Alors, pour son anniversaire, je lui ai construit un seul système à la place. Il s'appelle **Hearth**. Il lit chaque facture, chaque contrat et chaque reçu avec l'IA, les classe sous la bonne maison, la bonne voiture ou le bon projet, gère le calendrier, les notes et les mots de passe, et affiche le tout sur le web, sur son iPhone et son iPad, sur l'Apple TV du salon, sur le Mac et dans Chrome. Du premier prompt à la dernière version, il a fallu douze jours.

Cet article est une visite guidée, et c'est aussi un plaidoyer. À mon avis, la question « quel produit devrions-nous acheter ? » est désormais très souvent la mauvaise première question. Pour un nombre croissant d'entreprises, la meilleure question est « à quoi ressemblerait un logiciel taillé exactement pour nous ? » -- parce que le prix de la réponse s'est effondré. Je vais vous montrer ce qui a été construit, combien de temps cela aurait pris sans IA, où mon argument tient et où il ne tient pas, et ce que tout cela signifie si vous avez une application Delphi de bureau qui mérite une seconde vie.

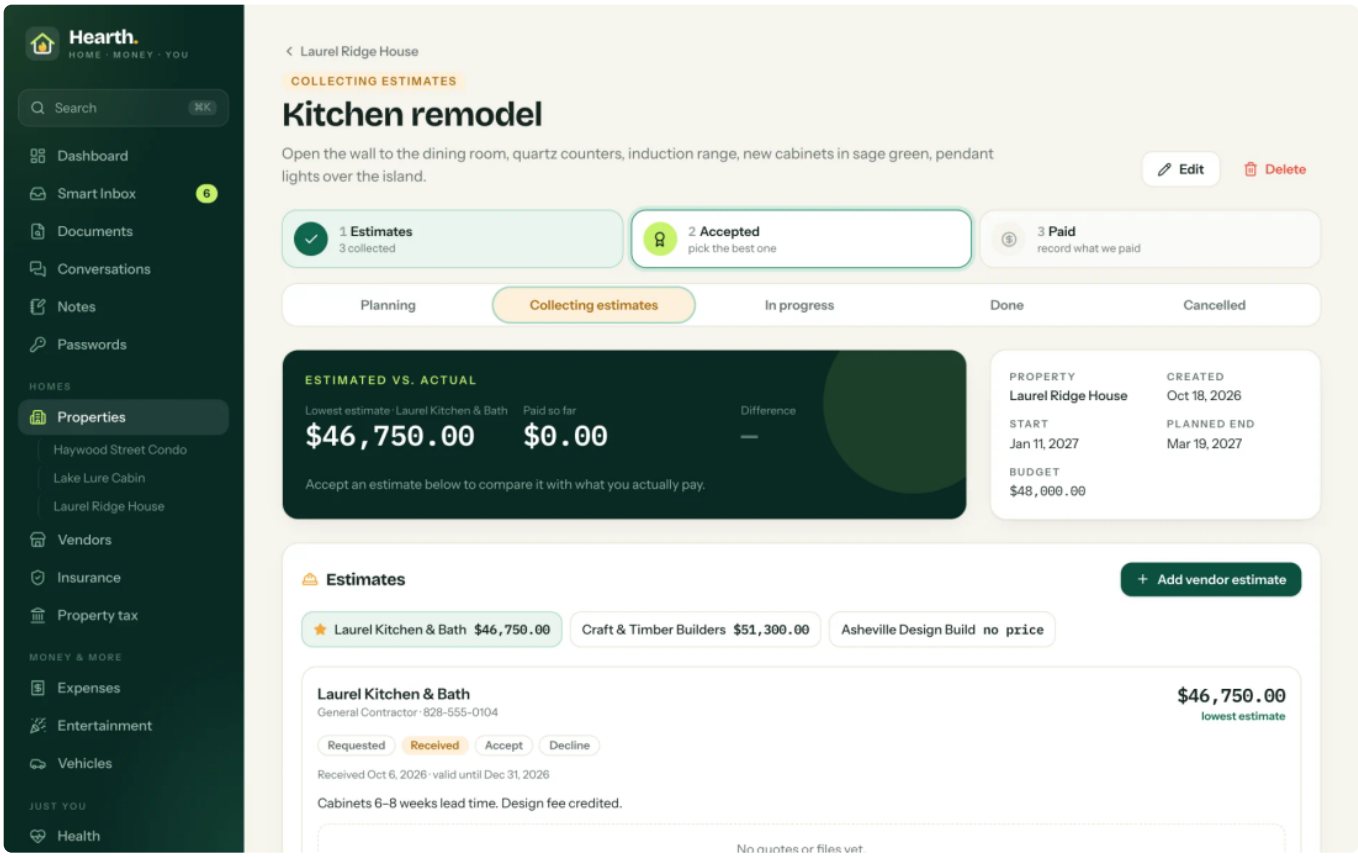
Hearth en un seul écran

Hearth gère un foyer de plusieurs maisons et voitures comme une petite entreprise gère ses opérations. Chaque journée commence sur le tableau de bord : la météo (avec une mention pour les jours où la lumière est bonne pour les photos d'annonces), les projets en cours, les dépenses de l'année, et tout ce qui attend une décision.



Le tableau de bord d'un foyer de démonstration fictif : la météo avec les jours de belle lumière pour les photos, les projets en cours et les dépenses de l'année.

Derrière le tableau de bord se trouvent les éléments qui étaient autrefois des outils ou des dossiers séparés. Les biens immobiliers viennent avec leur association de propriétaires (HOA), les avis de taxe foncière et leurs échéances, et les projets avec plusieurs devis d'artisans chacun. Les véhicules viennent avec leurs contrats de leasing, les assurances avec leurs périodes de couverture, leurs échéanciers de paiement et l'historique de chaque contrat, et l'assurance santé avec ses factures. Les fournisseurs ont leurs interlocuteurs et leurs numéros de téléphone, les conversations rassemblent e-mails et appels enregistrés, et les dépenses récurrentes se comptabilisent toutes seules. Les notes, les mots de passe et un calendrier complet parachèvent l'ensemble.

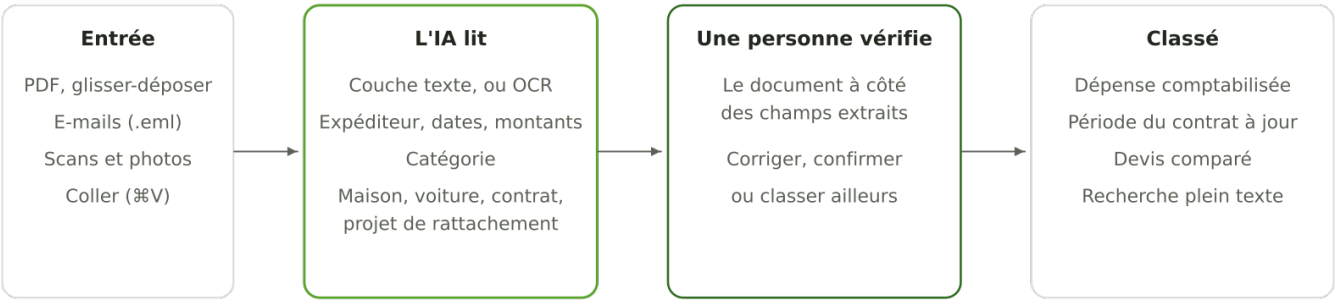


Un projet qui recueille des devis : trois entreprises côte à côte, la moins chère mise en évidence (données de démonstration).

Rien de tout cela n'est exotique. C'est précisément là où je veux en venir. Chaque fonction existe quelque part dans un produit commercial. Ce qui n'existe pas, c'est le produit unique qui a exactement ces fonctions, reliées exactement de cette façon, pour exactement cette personne.

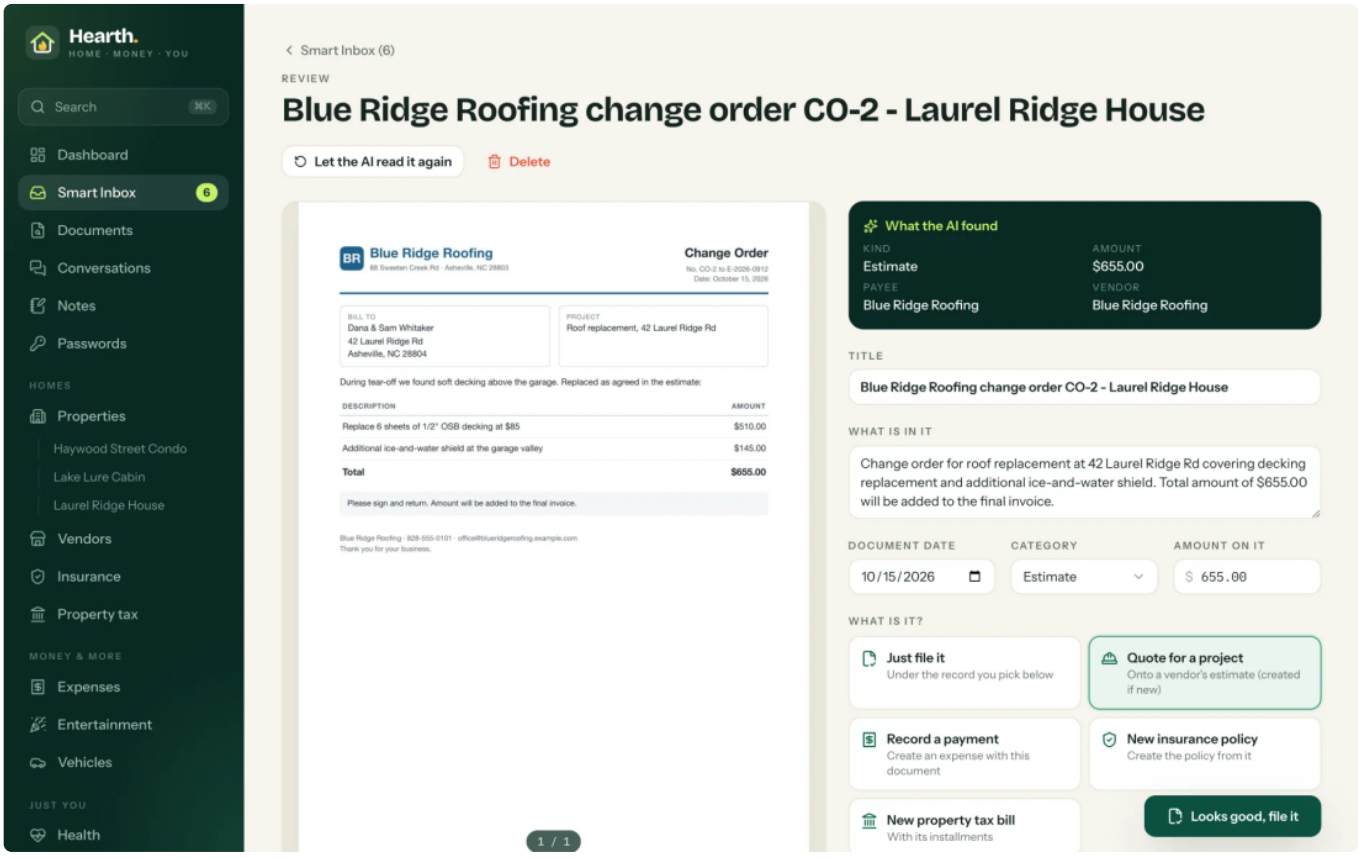
La pile de papier, réglée : OCR, catégories et classement

La fonction qui a changé le comptoir de la cuisine, c'est la chaîne de traitement des documents. Un document arrive sous forme de PDF, de photo, de scan ou d'e-mail enregistré avec ses pièces jointes, déposé sur la page ou collé avec V. Quand un PDF contient son propre texte, Hearth lit celui-ci ; quand il s'agit d'un scan ou d'une photo, chaque page passe par un modèle de vision qui fait office d'OCR. Ensuite, l'IA extrait l'essentiel : l'expéditeur, la date, le montant, l'échéance, le numéro de contrat ou de compte. Puis elle suggère où le document doit aller : quelle catégorie, quelle maison ou quelle voiture, quel contrat ou quel projet.



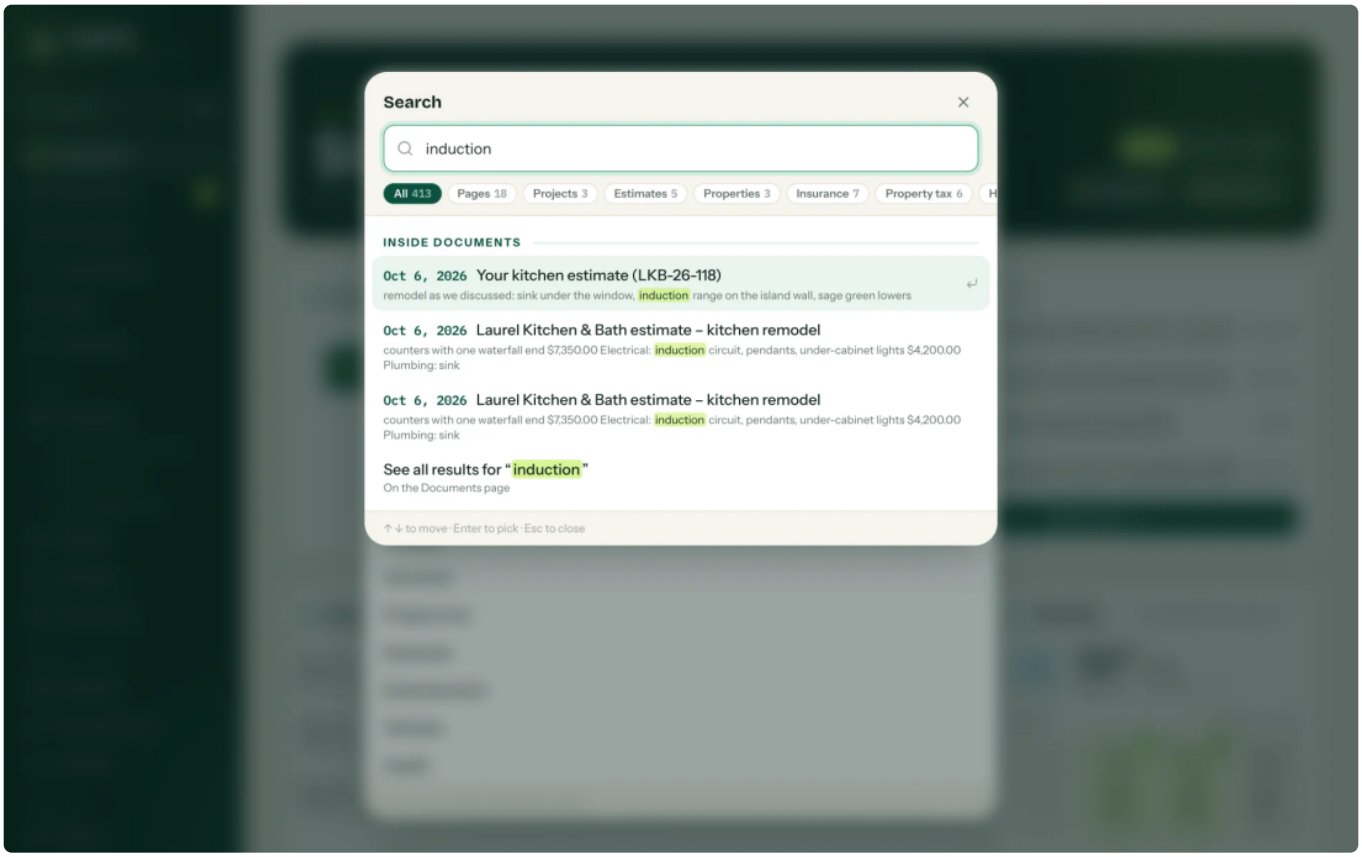
D'une pile de papier à un document classé et consultable -- avec un humain dans la boucle

Le troisième encadré est celui qui me tient le plus à cœur. L'IA ne décide jamais seule où va un document : tout ce qui arrive dans la Smart Inbox attend qu'une personne confirme ou corrige la suggestion. (Déposez un document directement sur un projet ou un contrat, et vous avez déjà dit où il va ; il saute donc la file d'attente.) L'écran de vérification affiche le document juste à côté de ce que l'IA a lu, si bien que la vérification prend quelques secondes au lieu de plusieurs minutes.



La Smart Inbox : le document d'un côté, de l'autre ce que l'IA a lu et où elle propose de le classer.

Une fois classé, un document travaille vraiment. Une facture devient une dépense sous la bonne maison, un renouvellement d'assurance prolonge le bon contrat, et le devis d'un artisan atterrit à côté des autres devis du même projet, si bien que l'offre actuelle la moins chère saute aux yeux. Chaque mot de chaque document se retrouve de partout avec K, et c'est généralement à ce moment-là que les gens arrêtent de me demander à quoi cela pourrait bien leur servir.



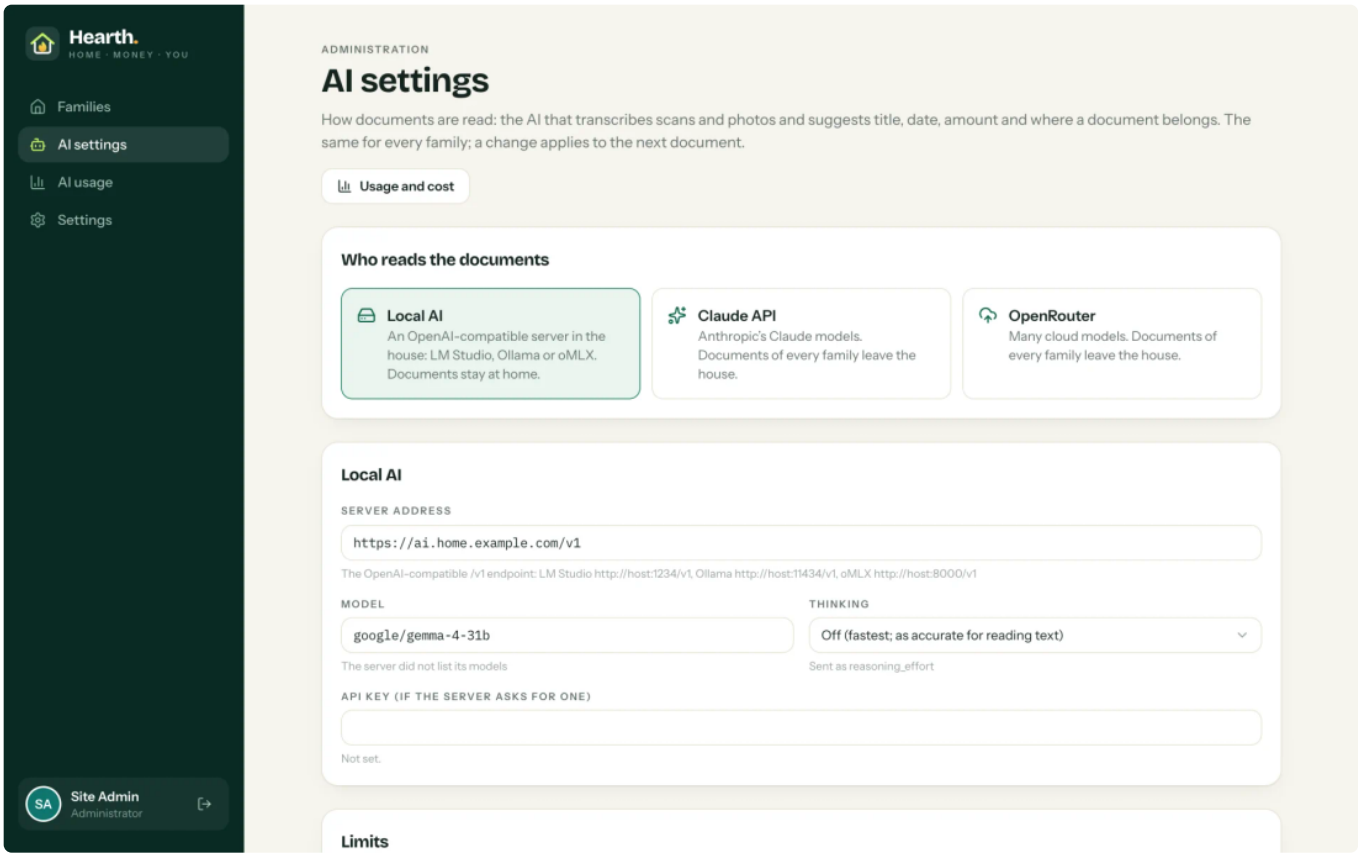
K recherche chaque mot de chaque document, où qu'il soit classé.

Les appels téléphoniques ont droit au même traitement. Après un appel avec le couvreur, touchez le micro, dites de quoi il était question, et Hearth le transcrit dans une conversation avec ce fournisseur. La transcription tourne sur un serveur Whisper installé sur un Mac de la maison, si bien que l'enregistrement ne quitte jamais le foyer.

N'importe quel fournisseur d'IA, y compris un qui vit à la maison

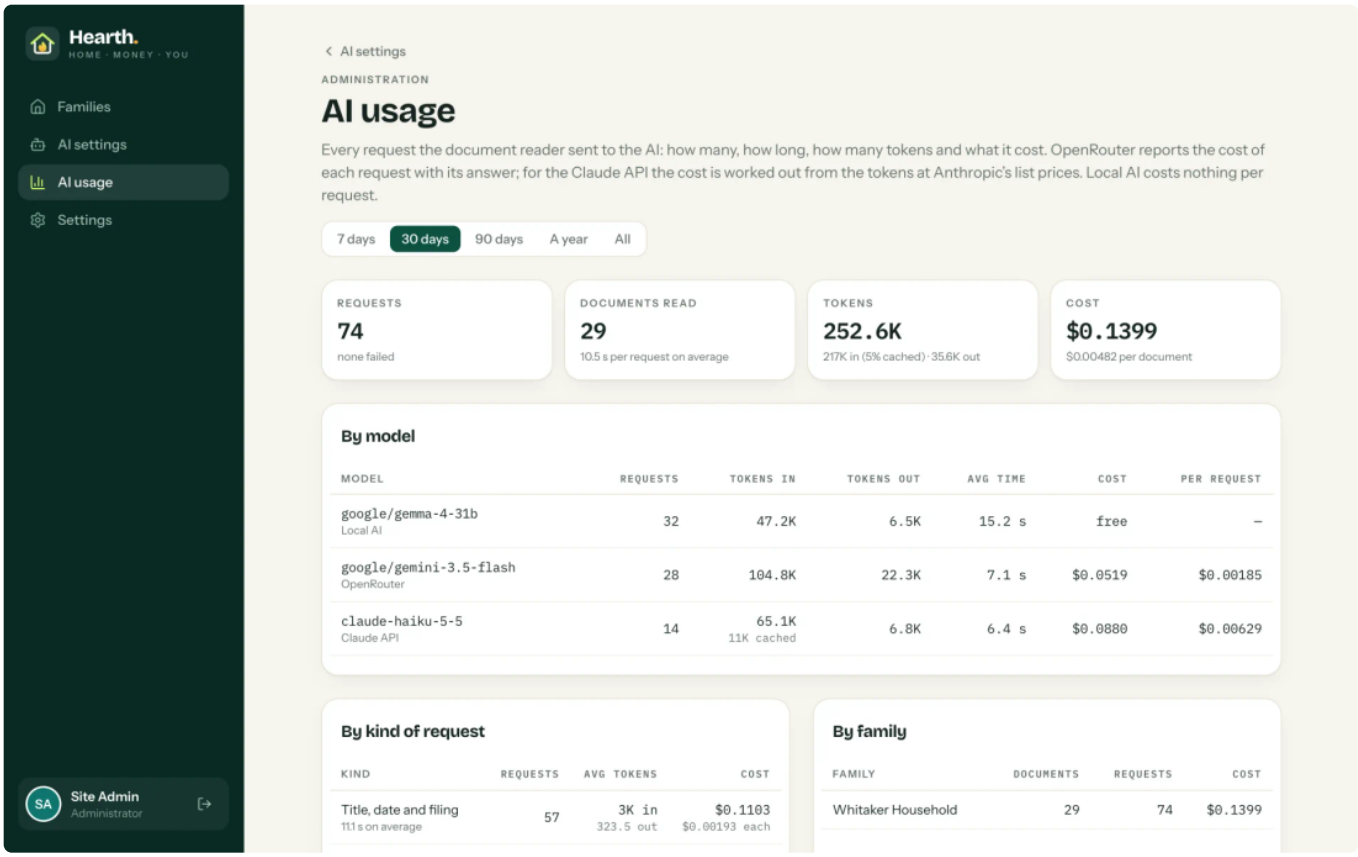
C'est le foyer qui choisit comment les documents sont lus, dans les réglages d'administration, sans nouvelle version. Hearth dialogue avec trois types de fournisseurs : l'API Claude d'Anthropic, n'importe quel modèle sur OpenRouter (qui réunit derrière une seule API les modèles d'OpenAI, de Google, d'Anthropic, de Mistral et de bien d'autres), ou un modèle local sur n'importe quel serveur compatible OpenAI, comme LM Studio ou Ollama.

Avec l'option locale, pas une seule page ne quitte la maison -- un sujet que j'ai traité en détail pour Delphi dans Give Your Delphi App a Brain.



Les réglages d'IA : Claude, n'importe quel modèle sur OpenRouter, ou un modèle local qui garde chaque page à la maison.

Parce que l'« IA » sans facture à la clé relève du conte de fées, chaque requête est journalisée avec ses tokens, sa durée, ses erreurs et son coût. Les modèles cloud indiquent ou estiment leur prix ; le modèle local ne coûte que l'électricité. L'administrateur voit tout cela par modèle, par type de requête, par famille et par jour.



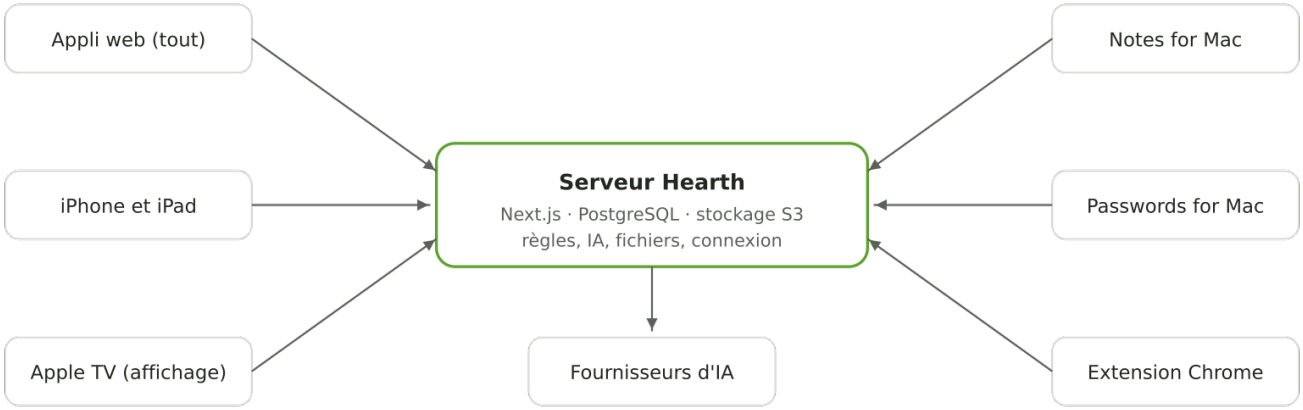
Utilisation et coût de l'IA : requêtes, tokens et coût par modèle et par fournisseur -- le modèle local ne coûte rien (données de démonstration).

Pourquoi la liberté de choisir son fournisseur compte pour votre produit

Si vous intégrez l'IA dans un produit aujourd'hui, le meilleur modèle de l'an prochain viendra très probablement d'un autre fournisseur que celui de cette année. Une application qui traite le fournisseur comme un réglage, et non comme une décision d'architecture, peut en changer en un après-midi. Elle permet aussi à un client soumis à des règles strictes de confidentialité de tout faire tourner en local, ce qui est un argument de vente à part entière.

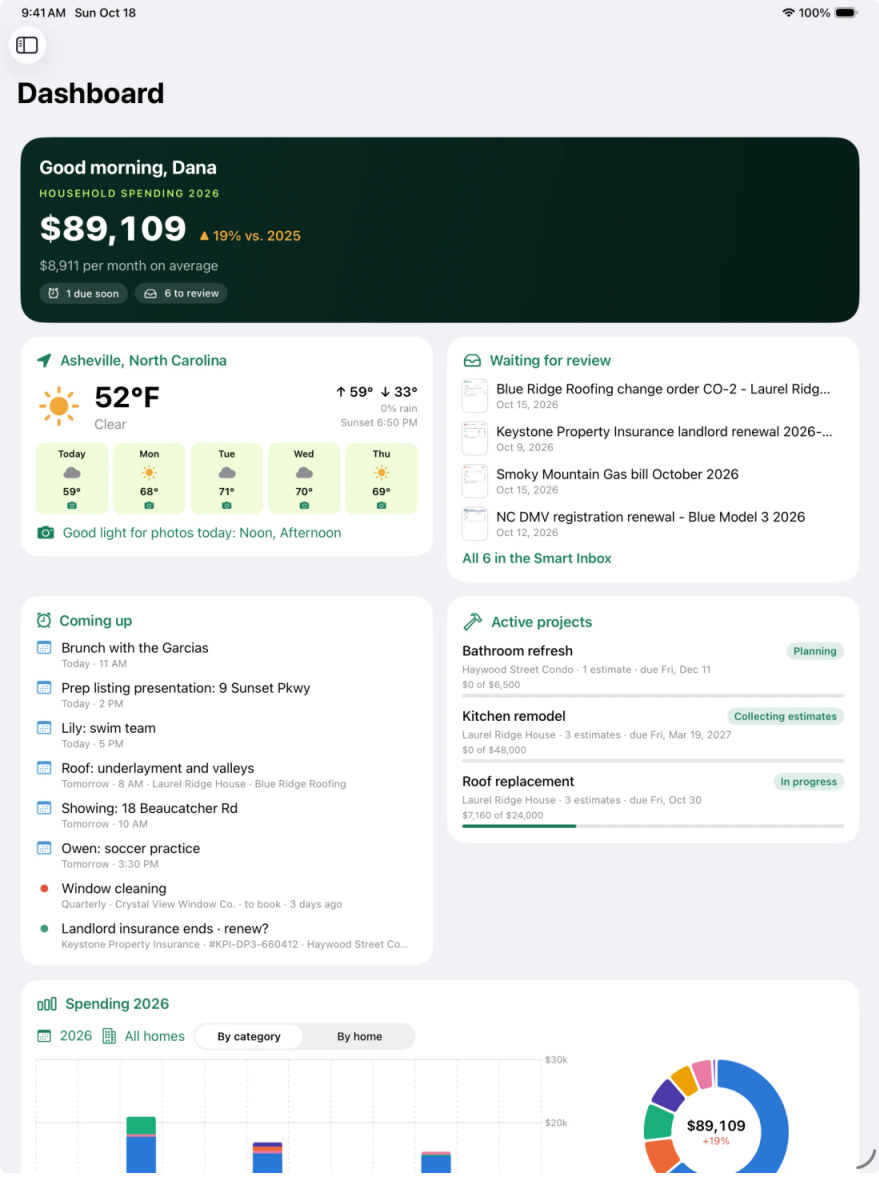
Un backend, sept portes d'entrée

Hearth n'est pas une appli ; c'est un serveur avec plusieurs portes d'entrée. L'application web fait tout. Une appli en lecture seule pour iPhone et iPad en affiche toutes les sections. Une appli pour Apple TV transforme l'écran du salon en tableau de bord familial. Notes for Mac et Passwords for Mac sont des applis natives complètes qui remplacent à part entière celles d'avant, et une extension Chrome remplit les identifiants directement sur les sites web.



Un serveur, une base de données, un seul jeu de règles -- et sept portes d'entrée

Le schéma paraît simple parce que la partie difficile est invisible : chaque porte d'entrée passe par les mêmes règles sur le serveur. Chaque appli se connecte séparément, avec un code confirmé sur le web, et chaque type d'appli ne peut faire que ce pour quoi il a été conçu. L'extension Chrome peut lire et enregistrer des mots de passe, et rien d'autre ; si quelqu'un volait sa connexion dans le navigateur, il ne pourrait pas ouvrir un seul document.

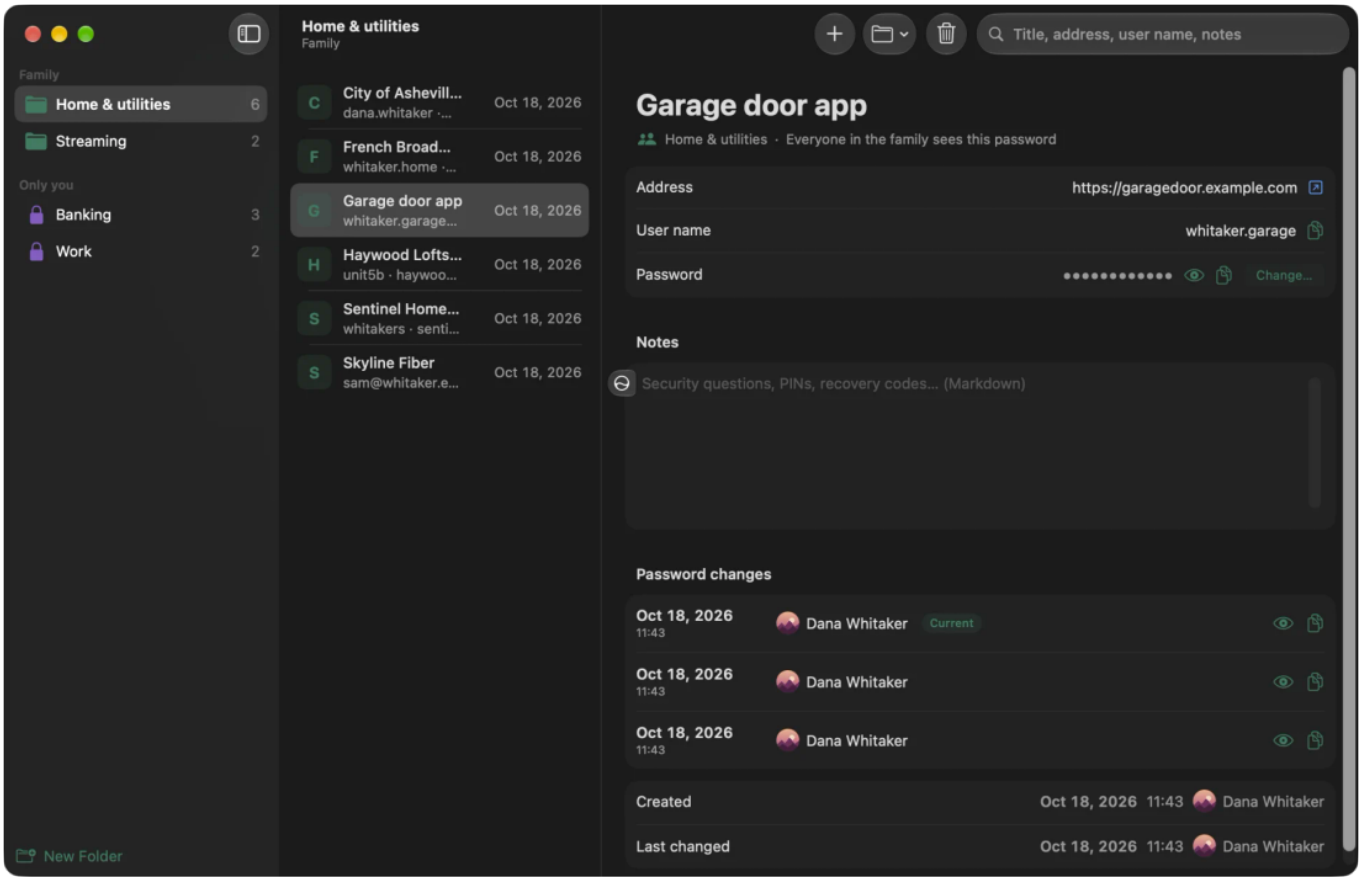


L'appli iPad affiche toutes les sections de Hearth, en lecture seule, avec les documents dans Coup d'œil.

L'appli Apple TV mérite qu'on s'y arrête, parce que c'est celle qui fait sourire. Elle affiche les rendez-vous du jour, le mois, la météo avec les heures de belle lumière pour les photos, et les dépenses de l'année, sous forme de diaporama qui tient l'économiseur d'écran à distance. Dans une maison, c'est un tableau de bord familial. Dans un bureau, un showroom ou une salle d'attente, exactement la même idée s'appelle **affichage dynamique**, alimenté en direct par vos propres données au lieu d'une clé USB que quelqu'un met à jour une fois par trimestre.

L'appli Apple TV : la journée, le calendrier, la météo et les dépenses -- un tableau de bord familial, ou de l'affichage dynamique dans un bureau.

Sur le Mac, les deux applis natives ont remplacé les applis de notes et de mots de passe que nous utilisions auparavant. Les mots de passe restent masqués derrière Touch ID et se verrouillent à nouveau au bout de cinq minutes, et l'extension Chrome propose d'enregistrer ou de mettre à jour un identifiant juste après que vous vous êtes connecté quelque part.

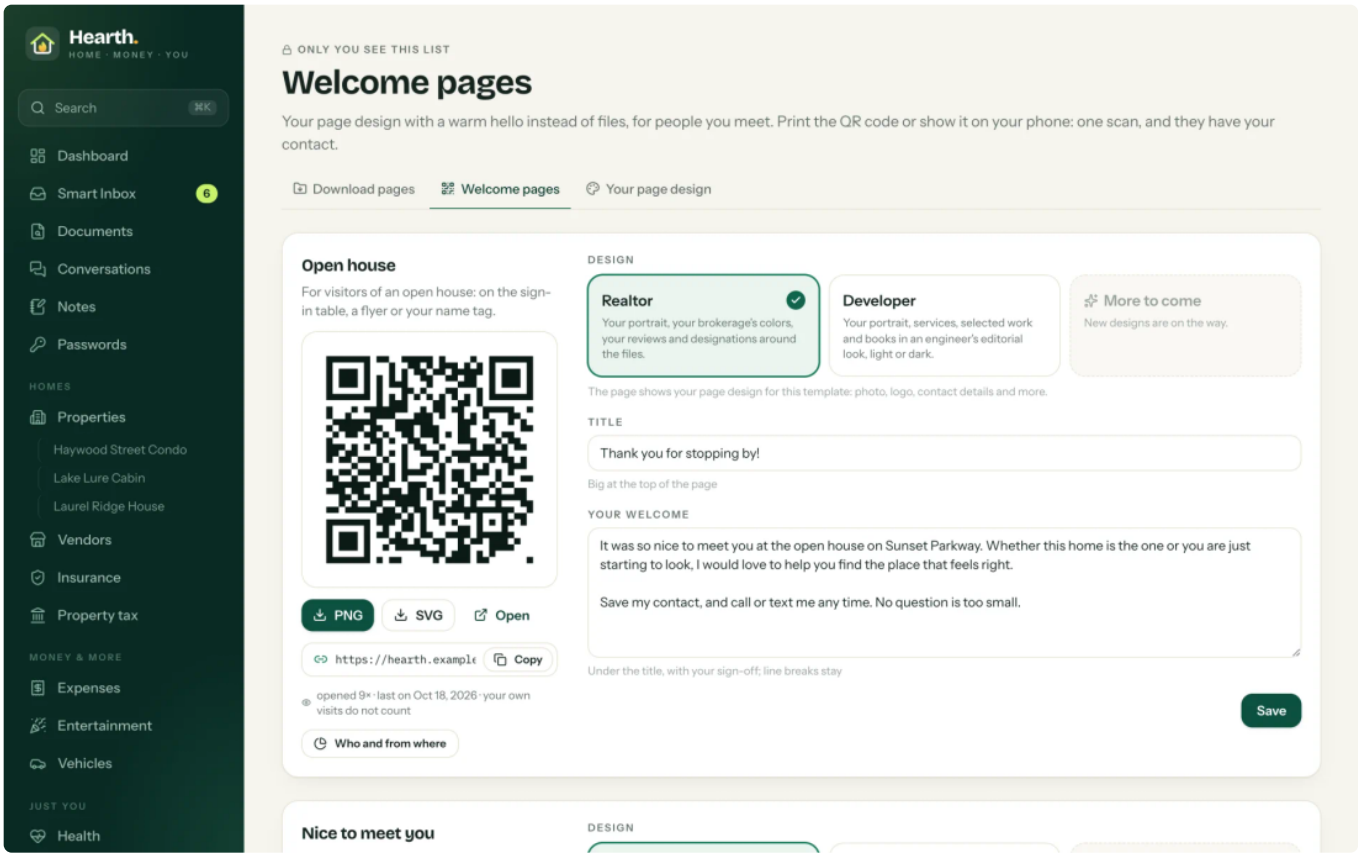


Passwords for Mac : verrouillé derrière Touch ID, avec l'historique des mots de passe et un générateur.

Conçu pour une agente immobilière, pas pour un million d'utilisateurs

C'est là que le logiciel sur mesure fait ses preuves. Un produit générique doit servir tout le monde, alors il ne sert exactement personne. Hearth sait que sa propriétaire vend des maisons, et il propose donc des choses qu'aucune appli domestique ne livrerait jamais.

Les pages de bienvenue en sont un bon exemple. Une agente immobilière peut créer une page pour un client ou une journée portes ouvertes -- à partir d'un modèle, avec un QR code à imprimer -- et Hearth enregistre quand elle a été ouverte, approximativement d'où, et avec quel navigateur. La carte météo signale les jours de belle lumière pour les photos d'annonces. Aucune de ces fonctions n'a pris longtemps à développer. Chacune serait une demande d'évolution dans le backlog d'un grand produit, soumise au vote de milliers d'utilisateurs qui n'en ont pas besoin.



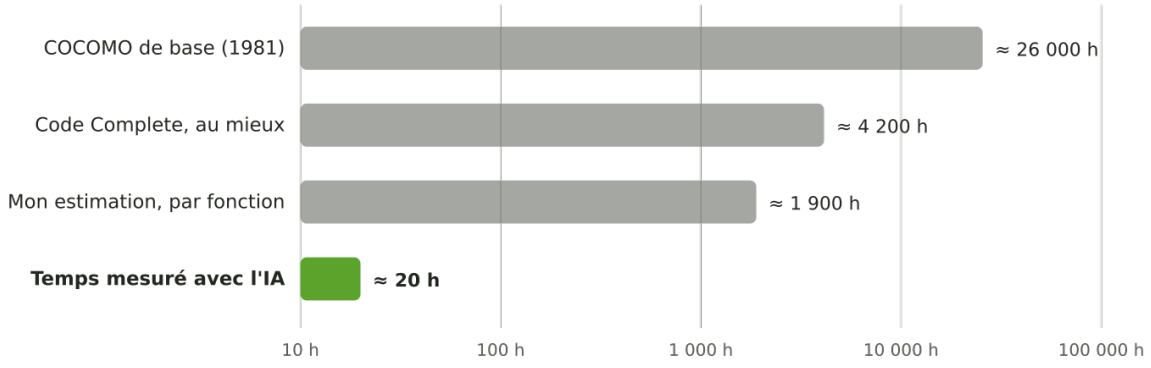
Une page de bienvenue avec QR code, créée à partir d'un modèle en une minute (données de démonstration).

> **Le même schéma, en entreprise**
> Remplacez « agente immobilière » par « courtier en assurances », « gestionnaire immobilier », « laboratoire de prothèses dentaires » ou « constructeur de machines », et l'histoire ne change pas. Une entreprise qui achète un grand CRM paie des centaines de fonctions qu'elle n'utilise jamais, puis paie encore -- en consultants ou en contournements -- pour les cinq dont elle a réellement besoin. Un système sur mesure part de ces cinq-là.

Combien de temps cela aurait-il pris sans IA ?

Douze jours, c'est la durée calendaire entre le premier prompt, le 7 octobre, et la dernière version, le 18 octobre. Les journaux de mes sessions Claude Code indiquent environ **20 heures** de travail actif avec l'IA sur cette période. Ce chiffre ne compte que le temps pendant lequel une session était occupée, les pauses de plus de vingt minutes étant exclues ; les tests sur l'iPad ou la réflexion sous le porche n'y figurent donc pas.

Le résultat : **58 588 lignes non vides** de TypeScript, Swift, SQL et JavaScript, 52 pull requests fusionnées et 48 versions taguées. Le nombre de lignes de code est une piètre mesure de la valeur -- Edsger Dijkstra soutenait, dans une formule restée célèbre, qu'il faudrait les comptabiliser comme « lignes dépensées » et non comme « lignes produites » -- mais c'est la seule mesure de taille qu'acceptent les modèles d'estimation classiques. Alors, posons-leur la question.



Effort estimé pour développer Hearth à la main, comparé au temps mesuré avec l'IA (heures, échelle logarithmique)

Le graphique a besoin d'une échelle logarithmique, ce qui en dit déjà long. Voici d'où vient chaque barre.

1. **COCOMO de base.** Le Constructive Cost Model de Barry Boehm, publié en 1981, estime l'effort à 2,4 × KLOC^1,05 personnes-mois pour un projet « organique ». Pour 58,6 milliers de lignes, cela donne environ 172 personnes-mois, soit à peu près 26 000 heures à raison des 152 heures par mois du modèle. Le modèle est ancien et suppose un monde sans frameworks modernes ; je le considère donc comme le plafond.
2. **Code Complete, au mieux.** *Code Complete* (2e édition) de Steve McConnell indique de 2 000 à 25 000 lignes par année-personne pour des projets de 10 000 lignes, et de 1 000 à 20 000 pour des projets de 100 000 lignes (tableau reproduit ici). En retenant l'hypothèse la plus optimiste, 25 000 lignes par an, Hearth représente environ 2,3 années-personnes, soit à peu près 4 200 heures à raison de 1 800 heures de travail par an.
3. **Ma propre estimation.** Je l'ai estimé comme j'estime les projets de mes clients : fonction par fonction, pour un développeur expérimenté, avec des frameworks modernes et sans IA. Le modèle de données et la connexion avec passkeys et isolation des familles : 25 jours. La chaîne de traitement des documents avec les fournisseurs d'IA : 25. Les modules du foyer, des biens immobiliers à l'assurance santé : 55. Calendrier, tableau de bord, recherche, notes, coffre-fort de mots de passe et pages de bienvenue : 44. Les tests, la revue de sécurité et le déploiement : 17. L'API des applis, l'appli iPhone et iPad, l'appli Apple TV, les deux applis Mac et l'extension Chrome : 69. Cela fait 235 jours ouvrés, soit un peu moins de 1 900 heures -- et c'est volontairement la plus optimiste des trois.

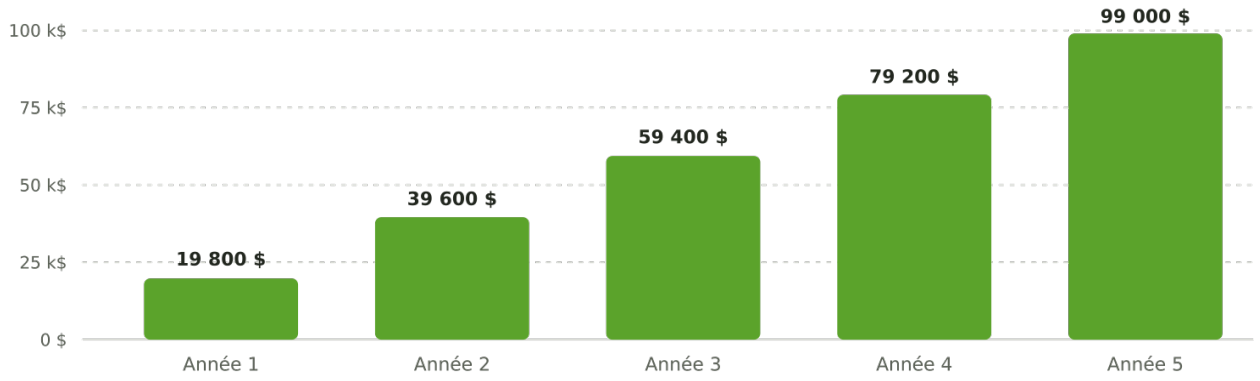
Même face à ma propre estimation optimiste, l'écart est d'environ deux ordres de grandeur. Je sais comment cela sonne ; la suite est donc le passage où je plaide contre moi-même.

> **Là où cette comparaison ne tient pas**
>
> Les 20 heures sont du temps de session, pas un décompte complet de ma journée : la réflexion, les tests sur les appareils et les décisions sur ce qu'il fallait construire se sont faits autour. L'IA a écrit bien plus de code que je n'en aurais tapé, et plus de lignes ne veut pas dire plus de valeur. Un projet parti de zéro, avec une seule utilisatrice exigeante et aucune donnée héritée, est le cas idéal pour l'IA -- une migration avec vingt ans de règles métier cachées dans une base de code Delphi 7 est une tout autre affaire, comme je l'ai décrit dans Adapt or Disappear. Et les études sont partagées : dans une étude randomisée de METR, des développeurs open source expérimentés ont été 19 % *plus lents* avec les outils d'IA du début 2025 sur leurs propres bases de code matures (METR souligne lui-même que ces résultats sont dépassés), et le rapport DORA 2024 a constaté que l'adoption de l'IA allait de pair avec une moindre stabilité des livraisons. Ma conclusion n'est pas « l'IA rend tout le monde 100 fois plus rapide ». Elle est plus étroite : un développeur expérimenté qui spécifie avec précision, relit tout et exige des tests peut désormais construire en quelques jours ce qui prenait autrefois un an.

Comment j'évite que cela tourne au « vibe coding » -- les tests, la revue de sécurité, la barrière avant chaque mise en production, les contrôles d'isolation --, c'est le sujet de mon prochain article, Pas du vibe coding.

Acheter ou développer ? La réponse honnête : « ça dépend »

Il existe un calcul simple que beaucoup d'entreprises ne font jamais. Un CRM de grand nom affiche son édition Enterprise à environ **165 à 175 $ par utilisateur et par mois**, selon le comparateur de prix consulté (Capterra, Redress Compliance). Pour dix utilisateurs, cela représente environ 20 000 $ par an rien qu'en licences, avant les modules complémentaires, les intégrations et le consultant qui le configure.



Licences cumulées au prix catalogue pour dix postes CRM à 165 $ par utilisateur et par mois (hors modules complémentaires et mise en œuvre)

Environ 100 000 $ sur cinq ans, pour un logiciel qui ne saura toujours pas ce que fait réellement votre entreprise. À mon avis, ce budget permet désormais de s'offrir un système sur mesure, avec de la marge -- un système qui vous appartient, qui parle votre langage, et qui grandit avec l'entreprise plutôt qu'avec la feuille de route de l'éditeur.

Cela dit, un abonnement achète de vraies choses qu'on oublie facilement : l'hébergement, les mises à jour, le travail de conformité, le support à 2 h du matin, et les intégrations avec un millier d'autres produits. Un logiciel sur mesure demande de la maintenance, des sauvegardes et quelqu'un qui répond quand il tombe en panne. Si votre processus est standard et qu'un produit y correspond, achetez le produit -- je le pense vraiment. Et si l'idée de posséder votre système vous plaît mais que vous ne voulez pas partir de zéro, il existe d'excellents CRM open source comme Odoo Community, SuiteCRM, EspoCRM et Twenty, que l'on peut personnaliser et héberger où l'on veut.

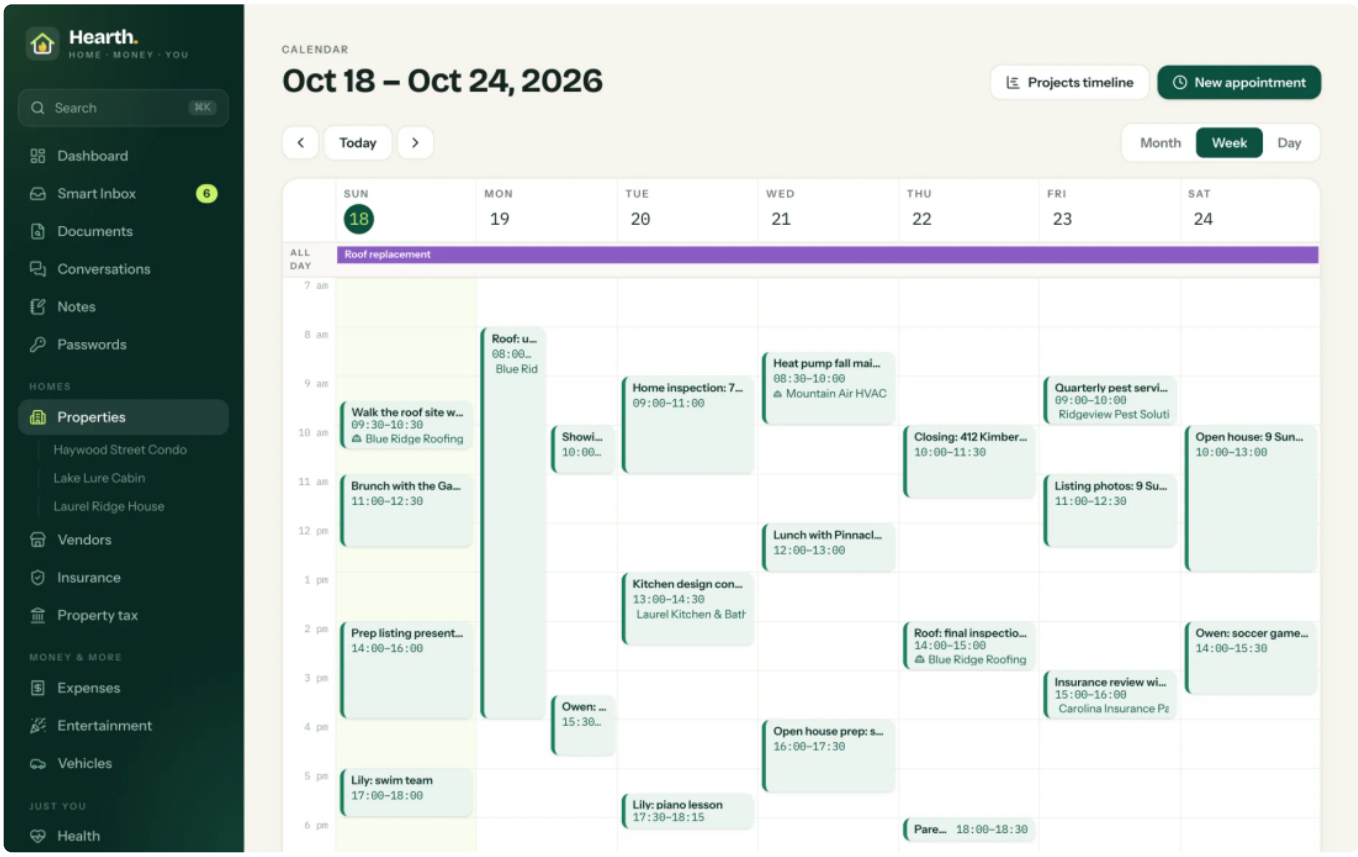
> **Quand développer**
> Développez quand le processus *est* votre avantage : quand vos équipes passent leurs journées à contourner un outil, quand les données sont éparpillées à cinq endroits, ou quand la seule fonction dont vous avez besoin est justement celle qu'aucun éditeur ne mettra jamais en priorité. Achetez quand le processus n'a rien de différenciant -- paie, messagerie, comptabilité -- et que le produit le fait déjà bien.

Ce que cela signifie pour votre application Delphi

La plupart de mes clients ne viennent pas me voir avec une page blanche. Ils viennent avec une application Delphi qui fait tourner leur entreprise depuis quinze ou vingt-cinq ans. Elle fonctionne, les utilisateurs l'adorent,

et elle devient peu à peu la raison pour laquelle l'entreprise ne peut rien entreprendre de nouveau : pas d'accès web, pas de smartphone, pas de tablette, pas d'IA, et un vivier de développeurs qui ne grossit pas.

Hearth donne une idée de ce que peut devenir une telle application, et il existe plusieurs chemins honnêtes pour y arriver. Vous pouvez conserver le code Delphi et placer une API REST devant -- avec TMS XData ou avec rien de plus que WebBroker -- puis construire par-dessus des interfaces web, mobiles et TV. Vous pouvez générer les pages web depuis Delphi lui-même avec WebStencils. Ou vous pouvez migrer l'ensemble vers une stack web moderne, le chemin que j'ai décrit dans ma série Delphi to Next.js. La bonne réponse dépend de votre code, de votre équipe et de vos utilisateurs, et j'aime mieux vous dire quel chemin vous convient que vous vendre celui que je préfère.



Le calendrier, avec des rendez-vous que l'on peut déplacer, redimensionner et ajouter d'un clic.

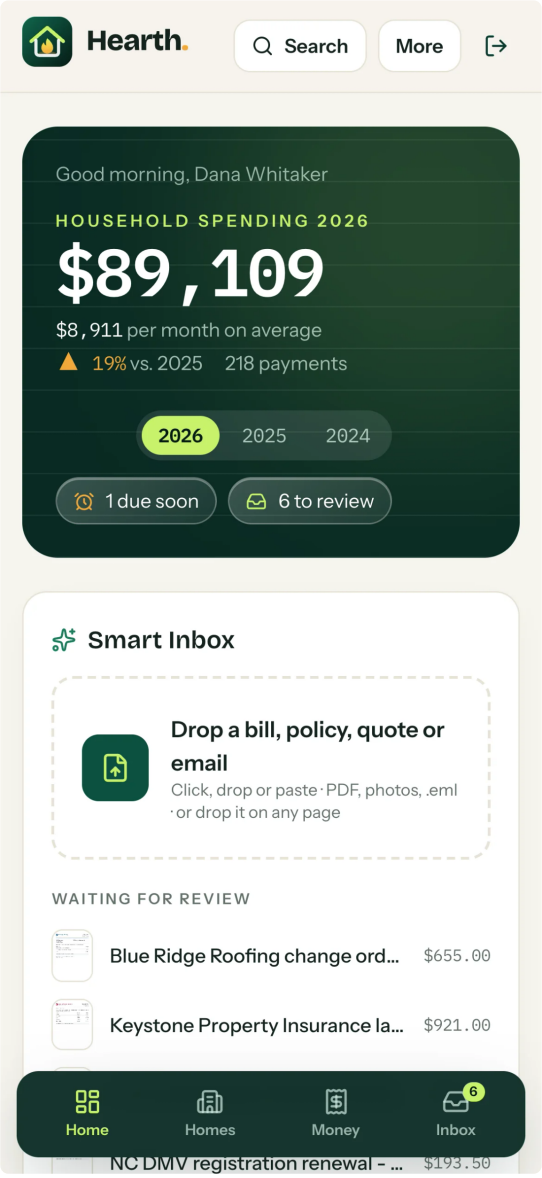
La modernisation d'un site web suit la même règle. Votre marque n'est pas un modèle tout fait ; je garde donc vos couleurs, votre typographie et votre logo, et je modernise ce qui se trouve en dessous : la vitesse, l'affichage sur smartphone, l'accessibilité, la recherche, et un circuit de publication qui n'exige pas un développeur pour chaque virgule. Si vous avez déjà un design system, nous travaillons à l'intérieur de celui-ci.

Équipes et développeurs solo

Je travaille des deux façons, et les deux sont courantes. Certains clients sont un développeur seul qui porte un produit depuis des années et qui a besoin d'un partenaire ayant déjà fait ce chemin -- quelqu'un pour relire le plan, prendre en charge les parties qui sortent de ses compétences, ou travailler en binôme sur le premier module. D'autres sont des équipes qui veulent la stack moderne et un processus assisté par IA qui fonctionne, introduits dans leur propre dépôt, avec leurs propres collaborateurs aux commandes et moi en relecture à leurs côtés. Dans les deux cas, le travail se fait dans un dépôt que vous voyez dès le premier jour, en versions courtes que vous pouvez faire tourner.

Nous commençons toujours de la même façon et, comme l'indique ma page de services, cela ne coûte rien :

une conversation de trente à soixante minutes sur le système et sur ce qui coince. Pour du code existant, l'étape suivante est un audit au périmètre fixe -- je le lis, je le compile, je l'exécute, et vous recevez un plan écrit et une estimation réaliste qui vous appartiennent, que nous continuions ensemble ou non. Vous trouverez d'autres systèmes que j'ai livrés dans la rubrique Work.



L'appli web sur un smartphone : le même système, mis en page pour le petit écran.

En résumé

Hearth a commencé comme un cadeau d'anniversaire et a fini par devenir le meilleur argument que je puisse avancer en faveur du logiciel sur mesure. En douze jours, un développeur expérimenté épaulé par l'IA a construit ce que les manuels situent entre une et plusieurs années-personnes : une IA qui lit et classe les documents avec n'importe quel fournisseur, des passkeys et un coffre-fort chiffré, et des applis natives pour iPhone, iPad, Apple TV et Mac, plus Chrome. L'IA n'a pas pris les décisions ; elle a rendu leur mise en œuvre bon marché.

> La question n'est plus de savoir si vous pouvez vous offrir un logiciel sur mesure. C'est de savoir si vous pouvez vous permettre de continuer à plier votre entreprise au logiciel de quelqu'un d'autre.

Si vous avez une application Delphi qui mérite un avenir, ou un processus auquel aucun produit ne correspond, j'aimerais beaucoup en entendre parler. Parlons-en.

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 —