$ npm run dev

npm run dev

Diari d'algú que aprén a crear aplicacions web amb IA

Samuel Soriano

npm run dev

Samuel Soriano

Índice

Crèdits 11

Introducció: `localhost` — allò funcionava i jo no sabia per què 11

`New Project` — la maqueta que entrava pels ulls 16

Apunt tècnic 25

Pàgina 25

Prototip 25

Frontend 26

Backend 26

Base de dades 26

Una pregunta millor per a començar . 27

`index.html` — un carrer per on podia caminar 28

Apunt tècnic 35

HTML: l'estructura 35

CSS: l'aspecte 36

JavaScript: el comportament 36

Quan començar amb un sol arxiu . . . 37

Un bon prompt per a aprendre 37

`code .` — la ferreteria on quasi compre una radial 38

Apunt tècnic 46
Obrir la carpeta correcta 46
Zones bàsiques 47
Pregunta útil a la IA 47
`git init` — el dia que no podia tornar arrere . 48
Apunt tècnic 56
Amb VS Code 56
Amb el terminal 57
Quan fer commit treballant amb IA . . 58
Una frase útil per a l'agent 59
`npm install` — un aeroport darrere de l'armari 59
Apunt tècnic 67
`package.json` 67
`npm install` 68
`node_modules` 68
`package-lock.json` 68
Abans d'afegir una dependència . . . 69
`npm run dev` — la llum que s'apagava amb el terminal 69
Apunt tècnic 77
On està definit 77

Què és `localhost` 78
El servidor local no és internet 79
Parar el servidor 79
Errors útils 79
`Build failed` — quatre hores per una icona . 80
Com llegir un error de build 87
Prompt útil 87
`/clear` — l'arxiu que ja no existia 87
Apunt tècnic 91
Abans de netejar 92
Després de netejar 92
`prompt.md` — el dia que vaig deixar de demanar favors 92
Apunt tècnic 99
Plantilla mínima 99
Prompt d'encàrrec 100
`opencode` — l'agent que parlava anglés en el pitjor moment 100
Apunt tècnic 105
1. Inspecció 105
2. Pla 105

3. Edició i verificació 106
Regles útils 106
Encàrrecs millors 106
`codex` — cometes, comes i mala idea 107
Apunt tècnic 113
`git diff` — el «Crear recurso» que ningú havia demanat 114
Apunt tècnic 118
Amb VS Code 118
Amb el terminal 119
Prompt útil 119
`console.log()` — el botó que callava 120
Apunt tècnic 126
Què donar-li a la IA 127
`npm test` — dotze verds que no vigilaven res . 127
Apunt tècnic 132
Un bon test 132
La prova del sabotatge 133
`localStorage` — el recurs que va sobreviure a un F5 134
Apunt tècnic 140

Quan és útil 141
Límits 141
Exportar sense atrapar dades 141
`supabase start` — la fila que no vaig copiar . 142
Apunt tècnic 146
Taula mínima 147
No és només codi 147
`.env` — una barra baixa a les 23:40 147
Apunt tècnic 153
Protegir-lo 153
Classifica les variables 154
Buscar secrets abans de commit . . . 154
Si ja has pujat una clau 154
Checklist abans de desplegar 155
`responsive` — deu segons al pati 155
Apunt tècnic 159
`deploy` — una URL i el botó mig tallat 160
Apunt tècnic 166
`It works on my machine` — el meu ordinador era còmplice 167
Apunt tècnic 172

`README.md` — l'aparador amb la porta tancada . 173
Apunt tècnic 181
Estructura mínima d'un README . . 181
Una bona issue 182
Llicències habituals 183
`git push` — nou hores amb la porta oberta . . 184
Apunt tècnic 189
Senyals que un secret s'ha filtrat . . . 189
Passos, en este orde 190
La regla de les dues claus 191
`v1.0.0` — dir que no la setmana abans del curs 191
Apunt tècnic 197
`maintenance mode` — el diumenge d'Empar . . 199
Apunt tècnic 205
Triar què fer 205
Dependències 206
Revisar abans de firmar — el canvi que quasi accepte a cegues 207
Apunt tècnic 213
Mira primer el diff 213
Zones de risc en una app web 214

Prompt de revisió 214
Criteri per acceptar 214
Epíleg: `exit 0` — firmar el que no has escrit del tot 215
Xuleta ràpida 222
Comandos bàsics 223
Projecte 223
Git 223
npm 224
Diagnòstic 224
Prompts útils 225
Checklist abans de publicar 225

Crèdits

npm run dev

Diari d'algú que aprén a crear aplicacions web amb IA

Versió: 1.0.3

Data de compilació: 2026-07-22

ISBN edició en paper: 9798186475095

Introducció: `localhost` — allò funcionava i jo no sabia per què

Hi ha paraules que, quan les veus per primera vegada, pareixen més grans del que són.

`localhost` n'és una.

La primera vegada que una aplicació web em va aparéixer en el navegador amb una adreça d'eixes — `http://localhost:3000`— vaig tindre una sensació estranya. No era exactament orgull, perquè jo encara no havia fet gran cosa. Tampoc era por, perquè la pàgina estava allí, blanca, obedient, amb un botó que fins i tot canviava de color quan passava el ratolí per damunt.

Era una sensació més incòmoda: la sospita que allò funcionava sense que jo acabara d'entendre per què.

Venia de fer les paus amb el terminal. Havia aprés a no espantar-me quan una orde em respo-

nia amb sequedat, a llegir errors sense sentir-me insultat, a no escriure `sudo` com qui pega un colp damunt la taula. El terminal, amb totes les seues manies, tenia una honestedat brutal: si no trobava una cosa, ho deia; si no tenies permisos, ho deia; si t'equivocaves de carpeta, també ho deixava clar, encara que fora després d'haver esborrat mig món.

El desenvolupament web amb IA és una altra classe de vertigen.

Ací no sols li demanes a la màquina que faça una cosa. Li demanes que construïsca una altra màquina: una aplicació amb botons, pantalles, dades, estats, usuaris, permisos, errors que només apareixen quan publiques, dependències que canvien de versió i fitxers que tenen noms aparentment innocents com `package.json`, `vite.config.js` o `.env`.

I la IA, si li dones peu, ho fa.

Eixe és el problema.

Ho fa massa prompte.

Abans, si volies crear una aplicació web, havies de travessar una porta estreta. HTML, CSS, JavaScript, algun framework, Git, un servidor, una base de dades, desplegament. La porta era tan estreta que

molta gent ni tan sols s'acostava. Ara, amb un editor com VS Code i qualsevol agent de codi —Copilot, OpenCode, Codex o el que s'invente demà—, la porta s'ha fet ampla. Pots escriure una frase i, en pocs minuts, tindre una cosa que sembla una aplicació.

Però una porta ampla no elimina l'escala que ve després.

Este llibre naix ahí: en eixe espai rar entre "no hauria pogut fer açò fa uns anys" i "si no entenc res del que ha fet la IA, açò no és meu".

No és un manual de React. No és una guia definitiva de VS Code. No és un catàleg d'eines, perquè qualsevol catàleg d'IA escrit hui envellix abans que el pa de motle. Tampoc és un manifest contra la programació tradicional ni una promesa que ja no cal aprendre.

És un diari d'aprenentatge.

El relat d'algú que vol crear aplicacions web reals amb ajuda de la IA i que, pel camí, descobrix que el treball important no és escriure tots els caràcters del codi. El treball important és entendre què estàs demanant, revisar què t'han donat, provar-ho,

protegir-ho, publicar-ho i fer-te responsable quan falla.

La IA pot escriure una funció en segons.

Però no sap si eixa funció respecta la privacitat del teu alumnat.

Pot afegir una base de dades.

Però no sap si realment necessites una base de dades o si t'estàs complicant la vida per vanitat tècnica.

Pot desplegar una aplicació.

Però no estarà en la reunió de dilluns quan algú et diga que en el seu mòbil el botó no es veu.

Per això m'interessa este tema. No per la fantasia de crear programari sense saber res, sinó pel contrari: perquè la IA m'ha obligat a aprendre coses que abans hauria ajornat indefinidament. M'ha posat una aplicació funcionant davant dels ulls i, després, m'ha deixat una pregunta damunt la taula:

Ara que existix, què penses fer amb ella?

La resposta no pot ser només "usar-la" ni "demanar-li a la IA que l'arregle quan falle". Si no la revise, si no entenc almenys per on circulen les dades, què fa cada pantalla i quins arxius sostenen

el conjunt, l'aplicació encara no és del tot meua. És una cosa que ha aparegut al meu ordinador.

I això, curiosament, no m'allunya d'aprendre. M'hi acosta. Com més entenc el codi, encara que siga a trossos, millor pregunte. Ja no li dic a la IA "fes que vaja bé", sinó "este estat es perd en recarregar", "esta funció barreja validació i presentació", "este botó hauria d'estar desactivat fins que hi haja dades". El coneixement no competix amb la IA: li dona direcció.

Este llibre és la resposta lenta a eixa pregunta.

Comença en `localhost`, eixe lloc que no està en internet però ja sembla món. Passa per VS Code, Git, `npm install`, builds que fallen, agents que toquen massa arxius, converses que convé netejar, dades que cal guardar, claus que no haurien d'haver arribat a GitHub i desplegaments que funcionen en el meu ordinador però no en cap altre lloc.

I acaba, si tot va bé, en una idea senzilla: crear amb IA no és deixar de ser autor.

És aprendre a ser-ho d'una manera més vigilada.

Tot comença, com quasi tot en este ofici, amb una carpeta buida i massa entusiasme.

`New Project` — la maqueta que entrava pels ulls

La carpeta buida va durar poc. La primera aplicació web que vaig demanar a una IA tenia un defecte greu: era massa bona.

No bona en el sentit profund de la paraula. No bona com una ferramenta provada, accessible, mantinguda i pensada per a persones reals. Bona en el sentit perillós: entrava pels ulls.

Jo havia obert VS Code amb la solemnitat ridícula de qui obri una llibreta nova el primer dia de curs. Carpeta buida, terminal neta, el cursor esperant. La pantalla no tenia encara cap culpa.

La idea era molt modesta, o això em vaig dir per a enganyar-me: una aplicació per a guardar recursos d'aula. Títol, assignatura, nivell, enllaç, una etiqueta, potser una valoració ràpida. Una cosa per a no perdre entre marcadors del navegador, documents solts i eixes converses de Telegram on Empar, la companya d'Infantil, passa un recurs mag-

nífic que desapareix tres dies després entre fotos d'excursions i recordatoris de claustre.

I tenia una data, encara que aquella vesprada no l'haguera dita en veu alta: volia que Empar poguera obrir açò el primer dia del curs que ve. Setembre pareixia lluny. No ho era.

Vaig escriure el prompt amb una alegria imprudent:

> Crea una aplicación web para docentes donde pueda guardar recursos educativos por materia, nivel y etiquetas. Que tenga diseño moderno, buscador, filtros, modo oscuro y posibilidad de editar y borrar recursos.

La IA no va dubtar.

En pocs segons em va tornar una estructura de projecte amb components, estils, icones, targetes, un buscador elegant i un botó d'afegir recurs que pareixia tret d'una aplicació de pagament. Hi havia una barra lateral. Hi havia filtres. Hi havia un estat buit amb una frase amable. Hi havia, per descomptat, mode fosc, perquè en algun moment de la

història recent vam decidir que cap aplicació podia nàixer sense la seua versió nocturna, encara que fora per a apuntar deures de segon d'ESO.

Vaig copiar, vaig instal·lar, vaig arrancar.

```
npm install
npm run dev
```

Dos comandos que encara no entenia: un instal·lava peces, l'altre encenia la pàgina. Els vaig escriure amb la fe de qui seguix una recepta en un idioma estranger. (Tots dos tindran capítol propi més avant.)

I allí estava.

`http://localhost:5173`

La meua aplicació.

O això vaig pensar durant uns deu segons.

Vaig afegir un recurs de prova: "Mapa interactiu de climes". Vaig posar-li etiqueta "Geografia", nivell "1r ESO", un enllaç qualsevol i una nota. Vaig prémer guardar. La targeta va aparéixer amb una animació suau, com si l'aplicació haguera estat esperant tota la vida que jo li confiara un mapa de climes.

Després vaig recarregar la pàgina.

El recurs va desaparéixer.

No va ser un error dramàtic. No va aparéixer cap missatge roig. No es va trencar res. Simplement, la meua aplicació va tornar a estar buida, com si mai no haguérem compartit aquell moment geogràfic.

Vaig mirar el codi. O, més ben dit, vaig mirar una massa de fitxers que contenien codi. `src`, `components`, `App.jsx`, `package.json`, `vite.config.js`. Tot tenia noms raonables i, precisament per això, em feia més respecte. La IA havia construït una façana preciosa, però jo encara no sabia distingir les parets mestres del decorat.

Li vaig preguntar:

> ¿Por qué al recargar se borran los recursos?

La resposta va ser educada, quasi maternal:

> Porque actualmente los datos solo se guardan en el estado de React. Si quieres que persistan al recargar, podemos usar localStorage o una base de datos como Supabase.

Ah.

"Actualmente".

Eixa paraula feia molt de treball.

Actualment l'aplicació no guardava res. Actualment jo tenia un formulari que fingia guardar. Actualment havia confós una maqueta interactiva amb una ferramenta. Actualment la IA m'havia donat exactament el que jo havia demanat, i el problema era que jo no sabia què estava demanant.

No havia dit "les dades han de conservar-se quan tanque el navegador".

No havia dit "ha de poder usar-se en dos ordinadors".

No havia dit "no vull servidor encara".

No havia dit "explica'm abans quines parts tindrà l'aplicació".

Havia dit "crea una aplicación web", que és com dir "fes-me una casa" sense aclarir si vols una maqueta de cartró, una caseta de jardí, un pis amb escriptura pública o un edifici amb ascensor i comunitat de veïns.

La IA, en eixa ambigüitat, havia triat la versió més vistosa i més ràpida: una aplicació de demostra-

ció. Útil per a veure la idea. Perillosa si jo la confonia amb una aplicació acabada.

Vaig sentir una punxada d'orgull ferit, que sempre és la forma més sorollosa de la ignorància. Durant uns minuts vaig estar temptat de demanar-li que ho arreglara tot:

> Añade persistencia, base de datos, login de usuarios y despliegue.

La frase em va passar pel cap amb la naturalitat de qui demana un cafè.

I ahí vaig parar.

No per prudència innata. Per memòria muscular. El terminal m'havia ensenyat una cosa: quan no entens on estàs, no continues avançant més ràpid. Escriu `pwd`. Mira la carpeta. Entén el terreny.

En desenvolupament web, el meu `pwd` era una pregunta més bàsica:

> ¿Qué tipo de aplicación me acabas de crear exactamente? Explícamelo separando prototipo, frontend, almacenamiento y backend.

La resposta va ser menys lluïda que la interfície, però molt més valuosa. Em va explicar que tenia un frontend: la part que es veu i s'executa al navegador. Que les dades vivien en memòria mentre la pàgina estava oberta. Que no hi havia backend: cap servidor propi que rebera i guardara informació. Que tampoc hi havia base de dades. Que allò era un prototip funcional de pantalla, no una ferramenta preparada per a usar-la de veritat.

Per primera vegada en aquella vesprada, la paraula "aplicació" es va fer més concreta.

No era una sola cosa.

Era una suma de decisions.

On es veu? En el navegador.

On es guarda? De moment, en cap lloc durable.

Qui pot entrar? Qualsevol que tinga la pàgina oberta, però no hi ha usuaris.

Què passa si tanque? Es perd.

Què passa si publique? Depén de com publique.

Què passa si un company afegix un recurs des del seu ordinador? Res, perquè el meu ordinador no sap res del seu.

Aquella primera aplicació, que deu minuts abans m'havia paregut un miracle, es va convertir en una cosa millor: un mapa de mancances.

No la vaig esborrar.

Després vaig obrir un arxiu nou i vaig escriure una llista:

```
# Què és real i què falta

- Es veu una interfície usable. - Puc afegir
recursos mentre la pàgina està oberta.
- Es perd tot en recarregar. - No hi ha
usuaris. - No hi ha base de dades. - No està
publicada. - No he revisat accessibilitat. -
No he provat en mòbil.
```

Eixa llista va ser el primer document honest del projecte.

La IA havia creat el primer prototip. Jo havia començat a entendre'l.

I, per primera vegada, em va semblar que potser d'això anava tot: no de fer que la IA escriguera codi, sinó d'aprendre a mirar el codi sense deixar-me hipnotitzar per la pantalla bonica.

La primera decisió en eixa direcció seria estranya: demanar-li a la IA que fera la meua aplicació més tonta.

Apunt tècnic

Quan li demanes a una IA que cree una "aplicació web", convé separar cinc paraules que en una conversa normal solem barrejar.

Pàgina

Una pàgina web pot ser només un document que el navegador mostra. Pot tindre HTML, CSS i una mica de JavaScript. Si no guarda dades ni té lògica complexa, pot viure perfectament com un arxiu o un conjunt d'arxius estàtics.

Prototip

Un prototip és una versió per a provar una idea. Pot semblar acabat i, alhora, no estar preparat per a un ús real. Moltes IAs creen prototips molt convincents: pantalles que funcionen prou per a ensenyar

el concepte, però sense persistència, seguretat, proves ni manteniment.

Frontend

El frontend és la part que veu i executa el navegador: pantalles, botons, formularis, estils i molta de la interacció. En projectes moderns sol estar fet amb eines com React, Vue, Svelte o altres frameworks, però la idea bàsica és la mateixa: és la cara visible de l'aplicació.

Backend

El backend és la part que corre en un servidor o servici extern. Rep peticions, valida dades, parla amb una base de dades, aplica permisos i retorna respostes. No totes les aplicacions necessiten backend des del primer dia, però cal saber si en tens o no.

Base de dades

La base de dades és on la informació viu de manera persistent. Si afegixes un recurs i després de recarregar la pàgina continua allí, les dades s'han

guardat en algun lloc: `localStorage`, un fitxer, una base de dades remota, Supabase, Firebase o un backend propi.

Una pregunta millor per a començar

En lloc de:

> Crea una aplicación web para guardar recursos educativos.

pots demanar:

> Quiero crear una primera versión de una aplicación web para guardar recursos educativos. Antes de escribir código, explícame tres opciones:
>
> 1. Prototipo sin guardar datos. 2. Versión con localStorage. 3. Versión con base de datos y usuarios.
>
> Para cada una, dime qué puedo hacer, qué limitaciones tiene y qué archivos tocarías.

La diferència és important: en el primer prompt demanes resultat; en el segon demanes criteri

abans de resultat. Amb IA, això sol marcar la frontera entre construir i només deixar-se impressionar.

Aprendre a crear amb IA comença quan deixes de preguntar "què pot fer?" i comences a preguntar "què estic acceptant quan dic que sí?".

`index.html` — un carrer per on podia caminar

Quan una cosa sembla massa intel·ligent, a vegades convé tornar-la més tonta. Ho havia decidit la nit anterior, encara amb la pantalla bonica del prototip fent-me l'ullet: si no podia explicar la meua aplicació, no era meua.

La meua primera aplicació de recursos d'aula tenia targetes, filtres, mode fosc i una elegància sospitosa. També tenia un problema elemental: jo no sabia explicar-la sense obrir-la. Si algú m'haguera preguntat "què hi ha dins?", hauria contestat amb una d'eixes frases que només servixen per a tapar ignorància:

"És una app feta amb React."

Com si dir "React" fora una explicació.

La vesprada següent vaig fer una cosa que al principi em va semblar un pas enrere. Vaig demanar a la IA que oblidara tot el projecte anterior i em

construïra una versió ridículament simple en un sol arxiu.

> Haz una versión mínima de la aplicación de recursos educativos en un único archivo index.html. Sin React, sin npm, sin dependencias. Solo HTML, CSS y JavaScript. Quiero entender cada parte.

La IA va tardar molt menys del que jo havia tardat a assumir que ho necessitava.

Em va tornar un arxiu llarg, però no incomprensible. A dalt hi havia una estructura HTML: títol, formulari, llista. Després un bloc de CSS amb colors, marges i botons. Al final, un bloc de JavaScript amb unes quantes funcions: afegir recurs, pintar la llista, esborrar.

Per primera vegada, la cosa no semblava una ciutat vista des d'un avió, sinó un carrer pel qual podia caminar.

Vaig crear una carpeta nova:

```
mkdir recursos-minim
cd recursos-minim
nano index.html
```

(`nano` és un editor de text que viu dins del terminal. Podia haver usat qualsevol altre; era nostàlgia.)

Vaig pegar el codi, vaig guardar i vaig fer doble clic sobre l'arxiu.

El navegador el va obrir.

Sense `npm install`.

Sense `node_modules`.

Sense servidor de desenvolupament.

Sense cap cerimònia.

La pàgina era més lletja que el primer prototip, clar. No tenia animacions suaus ni icones finíssimes. El mode fosc havia desaparegut, cosa que, contra tot pronòstic, no va provocar cap catàstrofe social. Però quan escrivia un recurs i premia el botó, entenia més o menys què passava.

El formulari tenia camps.

El JavaScript llegia els valors.

Creava un objecte.

L'afegia a una llista.

I tornava a dibuixar la pantalla.

No era poca cosa.

Vaig començar a fer el que faig quan una explicació em pareix massa neta: trencar-la un poc.

Primer vaig canviar el text del botó.

```
<button type="submit">Guardar recurs</button>
```

El navegador ho va mostrar. Bé.

Després vaig canviar un color.

```
button {
  background: #2563eb;
}
```

El botó es va fer blau. Cap sorpresa, i justament per això em va agradar.

Després vaig buscar la funció que afegia recursos i vaig escriure un `console.log`.

```
console.log("Recurs nou:", recurs);
```

Vaig obrir la consola del navegador —eixe panell mig amagat que apareix prement F12—, vaig afegir un recurs i allí estava: l'objecte apareixia com una fitxa xicoteta, amb el seu títol, la seua matèria i la seua URL. No era només una targeta bonica. Era una dada que circulava.

Aquell `console.log` em va fer més feliç del que seria raonable confessar en públic.

La IA havia generat el codi, sí. Però jo havia tocat una cosa i havia vist la resposta. Havia mogut una

peça i el sistema havia reaccionat. Durant anys havia pensat que programar era escriure instruccions perfectes en un llenguatge hostil. Aquell dia ho vaig viure d'una manera més humil: programar també era establir una relació de causa i efecte amb una pàgina.

Canvie açò.

Passa allò.

No entenc esta funció.

Pregunte.

Prove.

Em vaig adonar que el problema del primer prototip no era que fora massa modern. Era que havia aparegut massa complet. Com un alumne que entrega una redacció impecable escrita per algú altre: potser el text és bo, però no saps on posar la primera correcció.

Amb `index.html`, en canvi, tot estava al mateix lloc. Això té límits, evidentment. Una aplicació real no pot créixer indefinidament dins d'un sol arxiu sense convertir-se en una madeixa. Però com a primer mapa és magnífic. No perquè siga professional, sinó perquè és visible.

Vaig demanar a la IA que m'explicara el codi, però amb una condició:

> No me expliques todo el archivo de arriba abajo. Explícame qué ocurre cuando pulso el botón "Guardar recurs". Sigue el recorrido desde el formulario hasta que aparece la tarjeta.

La resposta va ser molt millor que una explicació general. Va seguir el camí del clic:

1. El formulari captura l'esdeveniment `submit`.
2. El codi evita que la pàgina es recarregue.
3. Llig els valors dels camps.
4. Crea un objecte amb eixes dades.
5. L'afegix a l'array de recursos.
6. Crida una funció que redibuixa la llista.

De sobte, el codi deixava de ser una paret i es convertia en una ruta.

Eixa va ser la primera lliçó real del llibre, encara que jo no ho sabia: quan treballes amb IA, no sempre has de demanar "fes-ho més potent". Moltes vegades has de demanar "fes-ho més simple fins que jo puga explicar-ho".

Perquè una aplicació que no pots explicar encara no és una ferramenta.

És un objecte trobat.

Ara tocava traslladar aquell carrer a un taller de veritat. I els tallers, ho descobriria prompte, tenen les seues pròpies temptacions.

Apunt tècnic

Un arxiu `index.html` pot contindre les tres peces bàsiques d'una pàgina web.

HTML: l'estructura

HTML diu què hi ha en la pàgina: títols, paràgrafs, formularis, botons, llistes, enllaços.

```
<form id="formulari">
  <input id="titol" placeholder="Títol del recurs">
  <button type="submit">Guardar recurs</button>
</form>
```

No diu com ha de ser de bonic ni què ha de passar quan prems el botó. Només dona estructura.

CSS: l'aspecte

CSS diu com es mostra eixa estructura: colors, espais, tipografia, columnes, grandàries.

```
button {
  background: #2563eb;
  color: white;
  border: 0;
  padding: 0.7rem 1rem;
}
```

Quan una IA genera una interfície molt vistosa, bona part de l'efecte està en el CSS. Això no és superficial: una app usable necessita disseny. Però el disseny no substituïx la lògica.

JavaScript: el comportament

JavaScript diu què passa quan l'usuari fa coses.

```
formulari.addEventListener("submit", function (event) {
  event.preventDefault();
  console.log("Formulari enviat");
});
```

Ací comença la interactivitat: llegir camps, guardar dades, canviar la pantalla, validar formularis.

Quan començar amb un sol arxiu

Té sentit començar amb `index.html` quan:

- vols entendre una idea abans de convertir-la en projecte gran;
- estàs prototipant una ferramenta senzilla;
- no necessites instal·lar dependències;
- vols enviar una demo que s'òbriga en qualsevol navegador;
- estàs aprenent i necessites veure-ho tot junt.

No té tant de sentit quan:

- l'aplicació té moltes pantalles;
- hi ha molts components repetits;
- necessites base de dades o usuaris;
- diverses persones treballen alhora;
- el fitxer comença a ser tan llarg que ja no trobes res.

Un bon prompt per a aprendre

> Crea una versión mínima en un único index.html. Después explícame el recorrido de una acción concreta: qué pasa desde que el usuario pulsa el botón hasta

> que la pantalla cambia. No quiero una explicación general, quiero seguir el camino de los datos.

Eixa última frase és clau: seguir el camí de les dades. En una aplicació web, entendre per on viatja la informació sol ser més útil que memoritzar noms de tecnologies.

El primer domini real no és dominar una eina. És poder seguir una acció senzilla sense perdre't: d'on ve, què toca i què deixa canviat quan passa.

`code .` — la ferreteria on quasi compre una radial

La primera vegada que vaig obrir VS Code per a un projecte web de veritat, vaig sentir que havia entrat en una ferreteria sense llista de la compra.

Tot pareixia útil.

Tot pareixia imprescindible.

I jo només volia penjar un prestatge sense acabar comprant una radial.

Venia del meu `index.html` minimalista, que tenia una virtut meravellosa: cabia sencer en el cap, o almenys en la part del cap que jo estava disposat a dedicar-li després de sopar. Però el món real no tardà a reclamar complexitat. La IA em deia que, si volia continuar amb l'aplicació de recursos d'aula, seria millor usar una estructura de projecte moderna. Vite, React, components, estils separats.

Vite i React encara no em deien res. De moment em vaig quedar amb la traducció menys solemne: ferramentes per a muntar la web per peces.

La paraula “moderna” sempre em fa desconfiar un poc. Sovint significa “funciona millor”, però també pot significar “necessita huit capes de ferramenta perquè un botó diga hola”.

Vaig crear una carpeta nova i la vaig obrir amb:

```
code .
```

El punt final em va semblar una broma privada del terminal. `code` era l’orde. `.` volia dir “ací”. Obri VS Code en esta carpeta, no en una idea abstracta, no en un núvol de possibilitats: ací.

La finestra es va obrir i, de colp, el meu projecte tenia paisatge.

A l’esquerra, l’explorador d’arxius. Al centre, l’editor. A baix, la terminal integrada. En un lateral, icones per a buscar, Git, extensions, xat. Era molt, però no era caos. Era una taula de treball amb massa calaixos.

La IA em va proposar instal·lar una extensió. Després una altra. Abans que em proposara la tercera, vaig fer la pregunta que em semblava més responsable:

> ¿Qué extensiones necesito para trabajar en este proyecto?

La resposta va ser un equipament d'expedició polar:

> Te recomiendo instalar: Prettier (formateo automático), ESLint, snippets de React, Auto Import, Path Intellisense, GitLens, Error Lens, Material Icon Theme, Import Cost y Live Share por si colaboras con alguien. También te preparo un settings.json con formateo al guardar y organización automática de imports.

Deu extensions i un arxiu de configuració per a un projecte que tenia cinc arxius. Però la llista venia firmada per una intel·ligència artificial, i jo encara estava en eixa fase en què tot el que té nom propi pareix imprescindible. Ho vaig instal·lar quasi tot i vaig acceptar el `settings.json` sencer.

Els problemes van tardar deu minuts. Vaig obrir `App.jsx` per a canviar un text, vaig guardar, i el formatador va reescriure l'arxiu de dalt a baix: cometes canviades, línies partides, sagnats nous. El

meu canvi d'una paraula s'havia dissolt dins d'un centenar de línies remogudes que jo no havia tocat. I al guardat següent, l'organitzador automàtic d'imports va decidir que una línia "no s'usava" i la va eliminar. La pàgina es va quedar en blanc.

Vaig passar mitja hora desfent la millora: desinstal·lar set extensions, desactivar el formatat en guardar, tornar a escriure a mà l'import assassinat. La IA no m'havia enganyat; m'havia equipat per a una expedició que jo no anava a fer. Vaig notar la mateixa temptació que quan un descobreix una botiga d'aplicacions: pensar que preparar l'entorn és treballar.

No sempre ho és.

De vegades és només procrastinar amb bona il·luminació.

Vaig decidir una norma: cap extensió que no poguera explicar en una frase. Si no sabia per a què servia, no entrava. Em vaig quedar amb dues.

Després vaig mirar la terminal integrada. Em va fer gràcia descobrir que, dins d'aquell editor tan ple de botons, el lloc on jo em sentia més tranquil era una línia negra esperant comandos.

```
ls
```

Allí estaven els arxius.

```
npm run dev
```

Allí arrancava la pàgina.

VS Code deixà de semblar-me una nau espacial quan vaig entendre que no era una alternativa al terminal, sinó una manera d'ajuntar peces que abans tenia escampades: arxius, terminal, Git, cerca i conversa amb la IA.

El moment important va arribar quan vaig obrir el panell de xat i li vaig demanar:

> Explícame esta carpeta como si fuera un mapa. No me digas qué es React en general. Dime para qué sirve cada archivo de este proyecto concreto.

La resposta va canviar la meua relació amb l'editor.

`package.json` no era "un fitxer raro de npm": era el lloc on el projecte declarava com arrancar i quines dependències necessitava.

`src/App.jsx` no era "el codi": era el component principal de la interfície.

`src/main.jsx` era l'entrada que connectava React amb el navegador.

`index.html` seguia existint, però ara era més com la porta d'entrada que com tota la casa.

`vite.config.js` era configuració de l'eina que feia possible el servidor local i el build, la versió empaquetada per a publicar.

El mapa no ho resolia tot, però reduïa el misteri. I reduir el misteri és una forma molt pràctica d'aprendre.

Durant uns dies vaig usar VS Code malament. Ho dic sense drama. Saltava d'arxiu en arxiu, acceptava suggeriments massa ràpid, obria tres xats diferents i després no sabia quin havia dit què. La IA, quan té accés al projecte, pot donar una sensació falsa de continuïtat: com si sempre recordara el que jo vull. Però no sempre ho recorda. De vegades només veu arxius i inferix una intenció.

Per això vaig començar a escriure notes dins del projecte.

Un `README.md` curt.

Un `TODO.md` encara més curt.

I, quan l'agent ho permetia, un arxiu d'instruccions amb normes bàsiques: no afegir dependències sense preguntar, no canviar l'estructura general sense explicar-ho, no tocar estils si l'encàrrec és de dades.

No era burocràcia. Era posar cartells en el taller.

El dia que VS Code va deixar de intimidar-me no va ser el dia que vaig aprendre tots els seus atajos. Encara hui ignore la meitat, i sospite que viuré dignament així. Va ser el dia que vaig entendre quines quatre zones necessitava per a treballar:

L'arbre d'arxius, per a saber on sóc.

L'editor, per a llegir i tocar.

La terminal, per a executar.

Git, per a no perdre'm del tot.

La IA era una cinquena zona, però no el centre. Un company de taula, no la taula.

Em va ajudar pensar en VS Code com pensava en una aula. Una aula pot tindre pissarra digital, ordinadors, projector, prestatgeries i murals. Però si no saps on seu cada alumne, on guardes els exàmens

i per quina porta entra la gent, tota la tecnologia només augmenta el soroll.

Un editor és igual.

No necessites venerar-lo.

Necessites orientar-te.

Orientar-me em va salvar de la ferreteria. No em va salvar del que va passar quan vaig deixar que algú reorganitzara el taller sencer sense haver fet abans ni una fotografia.

Apunt tècnic

VS Code és un editor, però en un projecte web sol actuar com a centre de treball. Estes són les peces que convé identificar prompte.

Obrir la carpeta correcta

Des del terminal:

```
cd ruta/del/projecte
code .
```

El punt (.) significa "la carpeta actual". És important obrir la carpeta del projecte, no un arxiu solt,

perquè així VS Code pot veure l'estructura completa, Git, dependències i configuració.

En macOS potser cal activar abans el comando `code` des de VS Code: `Shell Command: Install 'code' command in PATH`. En Windows, l'instal·lador sol oferir l'opció d'afegir VS Code al PATH.

Zones bàsiques

- **Explorer:** mostra arxius i carpetes.
- **Editor:** on lliges i edites fitxers.
- **Terminal integrada:** permet executar `npm`, `git` i altres comandos sense eixir de l'editor.
- **Source Control:** panell visual per a Git.
- **Search:** cerca text en tot el projecte.
- **Extensions:** afegits, però no cal instal·lar-ne massa.
- **Chat/agent IA:** ajuda contextual, segons l'eina instal·lada.

Pregunta útil a la IA

> Analiza esta carpeta como un mapa de proyecto. Explícame para qué sirve cada

> archivo principal y cuál es el flujo: qué archivo entra primero, dónde está la interfaz y qué comando arranca la aplicación.

Esta pregunta evita una explicació genèrica de la tecnologia i força la IA a aterrar en el teu projecte concret.

La productivitat no comença sabent cinquanta atajos. Comença sabent tornar a trobar el que acabes de trencar.

`git init` — el dia que no podia tornar arrere

El primer commit d'un projecte hauria d'arribar abans de la primera imprudència.

Jo, naturalment, ho vaig fer al revés.

Tenia l'aplicació de recursos d'aula oberta en VS Code. Ja no era aquell `index.html` tranquil del capítol anterior, sinó un projecte amb `src`, `components`, `package.json` i prou arxius perquè la confiança començara a necessitar casc. La interfície funcionava. Encara no guardava dades de manera seriosa, però ja podia afegir recursos, filtrar-los i veure'ls en targetes.

La IA em va proposar una millora:

> Puedo reorganizar el proyecto en componentes separados: ResourceForm, ResourceList, ResourceCard y Filters. También puedo preparar la estructura para añadir persistencia después.

Llegit així, semblava sensat. De fet, era sensat. El problema no era la proposta. El problema era la meua mà acostant-se massa ràpid al botó d'acceptar.

Hi ha un moment molt concret en què una persona deixa de dirigir la IA i comença a seguir-la. No sona cap alarma. No apareix cap finestra roja. Simplement, el text que et proposa pareix raonable, tu tens ganes d'avançar, i en lloc de preguntar "què tocaràs?", penses "avant, ja ho arreglarem si falla".

Eixa frase, "ja ho arreglarem si falla", és la porta d'entrada a moltes vesprades perdudes.

Vaig acceptar.

L'agent va començar a treballar. Va crear fitxers nous, va moure codi, va canviar imports, va netejar parts duplicades. Durant uns minuts em vaig sentir acompanyat per una eficiència quasi insultant. Jo hauria tardat una hora a fer aquella reorganització; ell la feia mentre jo mirava com apareixien noms nous a l'explorador.

Després vaig executar:

```
npm run dev
```

La pàgina arrancà.

Però el formulari no guardava.

No hi havia error visible. La consola mostrava un avís estrany sobre una propietat. Un botó havia canviat de lloc. Els filtres semblaven funcionar, però només a mitges. Vaig demanar a la IA que ho arreglara. Va tocar més fitxers. Ara el formulari guardava, però els filtres havien deixat de filtrar. Vaig demanar una altra correcció. Va tornar a tocar el codi. Ara hi havia dos components fent quasi el mateix.

Vaig mirar l'explorador de VS Code i vaig sentir una vergonya molt concreta: la de no saber tornar al punt anterior.

No havia fet commit.

Ni un.

Tenia una carpeta plena de canvis i cap fotografia del moment en què tot funcionava. Havia deixat que una IA reorganitzara la casa abans de fer una foto de com estaven els mobles.

En el panell lateral de VS Code, l'icona de Source Control tenia un número roig. Vaig fer clic. Allí hi havia una llista llarga de fitxers modificats, nous, suprimits. Tot barrejat. Sense cap punt de referèn-

cia. Vaig intentar donar-li sentit, però era com mirar les proves d'un judici on tots els testimonis parlen alhora.

Ni tan sols havia inicialitzat Git.

Vaig tindre la temptació de fer el que fem quan ens sentim culpables davant d'una màquina: escriure comandos sense pensar per a compensar. Però esta vegada vaig parar. Vaig copiar la carpeta sencera com a còpia bruta, amb un nom poc elegant:

```
cp -r recursos-aula recursos-aula-desastre
```

No era una solució professional. Era una tireta. Però em donava marge per a respirar.

Després vaig tornar al projecte, vaig demanar a la IA que m'ajudara a identificar una versió estable mínima i vaig desfer a mà part del soroll. No va ser heroic. Va ser lent, avorrit i necessari. Quan l'aplicació va tornar a funcionar en una versió acceptable, vaig fer el que hauria d'haver fet al principi.

Vaig obrir el panell de Source Control en VS Code. L'arxiu deia `git init` en una botó prominent: "Inicialitzar repositori". Vaig fer clic. De colp,

l'explorador va canviar d'aspecte. Els arxius que havia tocat apareixien marcats amb una "U" groga: nous, sense seguiment. Els que havia modificat tenien una "M" verda.

Vaig prémer "Etiquetar tots els canvis". Llista de fitxers va passar de "Canvis" a "Canvis preparats". Després vaig escriure un missatge en el quadre de text:

```
Versio inicial funcional
```

I vaig prémer la marca de verda de dalt.

El missatge no era poètic, però era una estaca clavada en terra.

A partir d'aquell moment, la relació amb la IA va canviar. No perquè la IA fora millor, sinó perquè jo tenia un punt de retorn. Li podia demanar:

> Refactoriza el formulario en un componente separado, pero no cambies el comportamiento. Toca solo los archivos necesarios.

I si la cosa es torcia, ja no depenia de la meua memòria ni de la bona voluntat del xat.

Podia mirar el panell de Source Control i veure exactament què havia canviat. Feia clic en un arxiu i apareixia el diff: línies verdes on s'havia afegit codi, línies rojes on s'havia llevat. Podia descartar mentalment una proposta abans d'acceptar-la. Podia tornar a començar.

El primer commit em va donar una cosa més important que seguretat tècnica: em va tornar l'autoritat. Abans del commit, cada canvi de la IA era una aposta. Després del commit, era un experiment.

La diferència és enorme.

Una aposta et posa nerviós perquè pots perdre el que tens.

Un experiment et permet aprendre perquè has protegit el punt de partida.

Des d'aquell dia vaig adoptar una norma senzilla: abans de demanar a una IA que reorganitze, refactoritze, instal·le, migre o "millore" qualsevol cosa que ja funciona, faig commit.

No sempre ho faig amb solemnitat. A vegades escric un missatge vulgar en el quadre de Source Control:

`Abans de tocar filtres`

I préme la marca de verda. Però funciona. El meu jo futur no necessita literatura. Necessita saber on estava el terra abans que començara l'obra.

També vaig aprendre una segona norma: un commit no és un contenidor de culpa. No cal esperar que el codi siga perfecte. Un commit pot dir simplement: "fins ací, açò arranca". En projectes amb IA, això val or, perquè els canvis poden ser ràpids, amplis i aparentment coherents fins que deixen de ser-ho.

Git no és només una ferramenta de programadors professionals.

Per a qui aprén amb IA, Git és el cinturó de seguretat.

I posar-se el cinturó no significa que conduïsques malament.

Significa que has entés què és moure's.

Amb el cinturó posat, ja podia mirar de cara l'altra font de vertigen del projecte: aquella carpeta que havia aparegut un dia i que pesava més que tota la resta junta.

Apunt tècnic

Git guarda la història del projecte. No cal dominar-lo sencer per a començar, però sí convé usar-ho des del primer dia.

Amb VS Code

El panell de Source Control de VS Code (l'icona de branca al lateral) permet fer les operacions bàsiques de Git sense tocar el terminal.

Inicialitzar Quan obrim un projecte que encara no té Git, el panell mostra un botó "Inicialitzar repositori". Fer clic ahí crea la carpeta oculta `.git` i activa el seguiment.

Veure què ha canviat Els arxius modificats apareixen a "Canvis". cadascun té una lletra que indica l'estat:

- **U** groga: arxiu nou, sense seguiment.
- **M** verda: arxiu modificat.
- **D** roja: arxiu suprimit.

Fent clic en un arxiu s'obri una vista de diff amb línies verdes (afegides) i rojes (eliminades).

Preparar i fer commit

1. Feu clic en "Etiquetar tots els canvis" o en el + al costat de cada arxiu individual.
2. Escriviu un missatge curt en el quadre de text.
3. Feu clic en la marca de verda de dalt, o premeu `Ctrl+Enter`.

El missatge ha d'ajudar-vos a recordar per què eixe punt era important. No cal que siga poètic. Només clar.

Descartar canvis Si un arxiu té canvis que no voleu conservar, podeu fer clic en el ~ al costat del nom per a tornar-lo a l'estat anterior.

Amb el terminal

Si preferiu el terminal, estos són els comandaments equivalents.

Inicialitzar:

```
git init
```

Veure l'estat:

```
git status
```

Preparar tots els canvis:

```
git add .
```

Fer un commit:

```
git commit -m "Versio inicial funcional"
```

Veure què ha canviat abans de fer commit:

```
git diff
```

Quan fer commit treballant amb IA

Fes commit abans de:

- demanar una refactorització;
- instal·lar dependències noves;
- canviar l'estructura de carpetes;
- tocar dades o persistència;
- acceptar una proposta gran d'un agent;
- començar una sessió llarga de proves.

Fes commit després de:

- aconseguir que una cosa funcione;
- arreglar un bug;
- completar una millora menuda;
- deixar el projecte en un estat que voldries poder recuperar.

Una frase útil per a l'agent

> Antes de hacer cambios, revisa el estado del proyecto. Propón un plan corto e indica qué archivos quieres tocar. No hagas refactors no relacionados con la tarea.

Git no impedix que la IA s'equivoque. El que fa és impedir que cada error siga una pèrdua de memòria.

`npm install` — un aeroport darrere de l'armari

El dia que vaig descobrir `node_modules` vaig pensar que havia trencat alguna llei de la física. Acabava d'estrenar el cinturó de seguretat de Git, i la primera cosa que em va demanar el cos va ser mirar per fi què hi havia dins d'aquella carpeta que sempre esquivava.

El projecte de recursos d'aula encara era menut. O això creia jo. Quatre components, uns estils, un formulari, una llista i la promesa de guardar dades algun dia d'una manera decent. La carpeta pareixia controlable. Fins i tot començava a reconéixer noms.

Llavors la IA em va dir:

> Antes de ejecutar el proyecto, instala las dependencias con npm install.

Dependències.

Una paraula elegant per a dir: “el teu projecte necessita coses que no són del tot teues”.

Vaig escriure:

```
npm install
```

I el terminal començà a parlar.

Línies i més línies. Paquets descarregats. Avisos que no sabia si eren greus o només adultesa tècnica. Noms que semblaven inventats per algú que havia perdut una aposta: `vite`, `eslint`, `postcss`, `nanoid`, `picocolors`. Jo només volia una aplicació per a guardar recursos educatius, però l’ordinador semblava estar muntant un festival.

Quan va acabar, vaig mirar la carpeta.

Hi havia aparegut `node_modules`.

Vaig entrar.

Error.

No d’ordinador. Meu.

`node_modules` no és una carpeta per a entrar a passejar. És una ciutat industrial. Milers d’arxius, carpetes dins de carpetes, paquets que depenen d’altres paquets que depenen d’altres paquets. Vaig

sentir una cosa molt semblant a obrir un armari i descobrir que darrere hi havia un aeroport.

Vaig preguntar a la IA:

> ¿Es normal que node_modules sea tan grande?

La resposta va ser tranquil·la, que és una manera molt fina de no compartir el teu pànic:

> Sí. node_modules contiene todas las dependencias necesarias para ejecutar el proyecto localmente. No debes editar esa carpeta ni subirla a Git.

No editar. No pujar.

Dues instruccions que vaig escriure mentalment en lletres grans.

Després vaig obrir `package.json`. A diferència de `node_modules`, sí que podia mirar-lo sense perdre la noció del temps. Tenia un aspecte més humà:

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build"
```

```
  },
  "dependencies": {
    "@vitejs/plugin-react": "latest",
    "vite": "latest",
    "react": "latest",
    "react-dom": "latest"
  }
}
```

Aquell arxiu era una mena de llista de la compra del projecte. Deia quines peces externes necessitava i quins comandos sabia executar. La carpeta `node_modules` era el rebost ple després d'haver anat al supermercat. I `package-lock.json` era el tiquet detallat que deixava constància de versions exactes, substitucions i tota la burocràcia invisible de la compra.

La metàfora em va servir, encara que no fora perfecta. Sobretot perquè m'ajudava a entendre una cosa important: jo no havia instal·lat "React" com qui instal·la una aplicació tancada. Havia descarregat una constel·lació de peces que el projecte necessitava per a arrancar.

I eixa constel·lació canvia.

Això em va incomodar.

Venia d'un món on un arxiu era un arxiu. El guardaves, el copiaves, l'obries. En el desenvolupament web modern, en canvi, un projecte pot ser més una recepta que un plat. Tu guardes els ingredients declarats, i quan algú executa `npm install`, el sistema reconstrueix la cuina.

Per això no puges `node_modules` a Git. No perquè siga il·legal ni pecat, sinó perquè és enorme, generat i reconstruïble. El que interessa guardar és la recepta: `package.json` i `package-lock.json`.

Vaig revisar el `.gitignore` —l'arxiu on s'apunta tot allò que Git ha de fer com si no veiera— i vaig comprovar que incloïa:

```
node_modules/
```

Vaig sentir una satisfacció molt domèstica. Com tancar bé una finestra abans d'anar a dormir.

Després arribaren els avisos.

```
added 187 packages, and audited 188 packages in 12s

12 packages are looking for funding
found 0 vulnerabilities
```

El primer dia vaig llegir “vulnerabilities” i em va pujar el pols. Després vaig entendre que npm aprofita cada instal·lació per a mirar si alguna dependència té problemes de seguretat coneguts. Quan diu `found 0 vulnerabilities`, respires. Quan no diu zero, tampoc cal incendiar el portàtil. Cal llegir, entendre si afecta el teu cas i actualitzar amb criteri.

La IA, com sempre, tenia una solució ràpida preparada:

> Puedes ejecutar npm audit fix.

I jo, ja una mica més escarmentat, vaig preguntar abans:

> ¿Qué cambiaría exactamente npm audit fix en este proyecto?

Eixa pregunta em va evitar més d’un disgust posterior. Algunes ordres que semblen de manteniment poden actualitzar versions, canviar el `package-lock.json` i trencar compatibilitat. No sempre, però prou vegades per a meréixer respecte.

Aquell dia no vaig aprendre a dominar npm. Això seria mentir. Vaig aprendre una cosa més modesta i

més útil: `npm install` no és una fórmula màgica per a "preparar la app". És el moment en què el projecte declara les seues dependències i tu acceptes que una part del teu treball descansa sobre codi escrit per altres.

La IA també escriu codi que no és del tot teu.

Les dependències també.

La diferència és que `package.json` almenys ho deixa per escrit.

Vaig fer commit. Vaig obrir el panell de Source Control en VS Code, vaig veure que `package.json`, `package-lock.json` i `.gitignore` hi eren com a canvis nous, i vaig escriure un missatge curt:

```
Afig dependències inicials
```

Després vaig prémer la marca de verda.

No vaig afegir `node_modules`.

Per una vegada, vaig tindre la sensació agradable d'haver no fet una cosa correctament.

Quedava per entendre l'altra meitat de la fórmula que repetia des del primer dia sense pensar: què passava exactament quan escrivia `npm run dev`.

Apunt tècnic

`npm` és el gestor de paquets més habitual en projectes JavaScript. Permet instal·lar dependències i executar scripts del projecte.

`package.json`

És un arxiu central. Sol contindre:

- nom del projecte;
- scripts disponibles;
- dependències necessàries;
- dependències de desenvolupament;
- metadades.

Exemple:

```
{
  "scripts": {
    "dev": "vite",
    "build": "vite build"
  },
  "dependencies": {
    "react": "^19.0.0",
    "react-dom": "^19.0.0"
  },
  "devDependencies": {
    "vite": "^7.0.0"
```

```
  }
}
```

npm install

Instal·la les dependències declarades en `package.json` i genera o actualitza `package-lock.json`.

```
npm install
```

Si baixes un projecte d'internet o el clones amb Git, normalment el primer pas és executar `npm install`.

node_modules

És la carpeta on npm descarrega les dependències. Pot ser molt gran. No l'has d'editar a mà i normalment no s'ha de pujar a Git.

Al `.gitignore`:

```
node_modules/
```

package-lock.json

Guarda versions concretes de les dependències instal·lades. Ajuda a reproduir el mateix entorn en altres ordinadors.

En general, sí que convé pujar-lo a Git en aplicacions.

Abans d'afegir una dependència

Pregunta a la IA:

> Antes de añadir una dependencia nueva, explícame: - para qué sirve; - si podemos hacerlo sin dependencia; - qué mantenimiento implica; - qué archivos cambiarán.

Una dependència pot estalviar faena, però també afegix pes, risc i obligacions. En un projecte xicotet, de vegades la millor dependència és la que no instal·les.

Cap projecte és del tot solitari. Inclús quan treballes a soles, portes dins el codi d'altres, decisions d'altres i manteniments que no controles. La maduresa no és evitar eixa xarxa; és saber quan entrar-hi i amb quin deute.

`npm run dev` — la llum que s'apagava amb el terminal

Hi ha un moment en què una aplicació deixa de ser una carpeta.

No és quan escrius el primer arxiu. Tampoc quan la IA et diu que tot està preparat. Ni tan sols quan VS Code deixa de mostrar errors de colors.

És quan escrius:

```
npm run dev
```

i el terminal respon amb una adreça.

> Local: http://localhost:5173/

La primera vegada que vaig veure eixa línia vaig sentir una alegria infantil. No perquè entenguera tot el que estava passant, sinó precisament perquè no ho entenia del tot i, malgrat això, allí hi havia una porta.

Vaig fer `Ctrl+clic` sobre l'enllaç.

El navegador s'obri.

L'aplicació apareix.

No com un arxiu HTML obert a soles, sinó com una cosa servida, viva, pendent de canvis. Vaig modificar un text en `App.jsx`, vaig guardar i la pàgina es va actualitzar quasi a l'instant.

Vaig canviar:

```
<h1>Recursos d'aula</h1>
```

per:

```
<h1>Biblioteca d'aula</h1>
```

I el navegador va obeir sense que jo recarregara res.

Hot reload, em va dir la IA.

Recarrega en calent.

Un nom molt dramàtic per a una comoditat molt simple: tocar codi i veure el canvi immediatament. Però eixa comoditat canvia la manera d'aprendre. Amb un cicle tan curt, proves més. Si proves més, preguntes millor. Si preguntes millor, la IA deixa de ser una font d'oracles i es convertix en una companya de laboratori.

Durant uns minuts vaig fer experiments absurds.

Canviar el títol.

Canviar el color d'un botó.

Trencar un component llevant un parèntesi.

Veure l'error.

Tornar-lo a posar.

Mai no havia valorat tant un error immediat. Quan l'aplicació fallava en guardar, el navegador o el terminal m'ho deien. A diferència d'altres errors de la vida, este venia amb línia aproximada i nom d'arxiu.

Però l'eufòria local té trampa.

Vaig tancar el terminal.

La pàgina del navegador continuava oberta, però ja no responia igual. Vaig recarregar i aparegué un missatge d'error. La meua aplicació no vivia "en el navegador" d'una manera autònoma. Depenia d'un procés que estava corrent en el meu ordinador.

El servidor de desenvolupament era com una cafeteria provisional muntada dins de casa: mentre la cafetera està encesa, pots servir cafés. Quan l'apagues, no hi ha negoci.

Vaig preguntar:

> ¿Entonces mi aplicación ya está en internet?

La IA respongué amb una paciència que vaig agrair:

> No. Está funcionando en tu ordenador. localhost significa esta máquina. Para que otras personas puedan acceder, necesitas desplegarla en un hosting.

`localhost` vol dir esta màquina.

No “el meu lloc web”.

No “la meua aplicació publicada”.

No “internet, però xicotet”.

Esta màquina.

Vaig recordar la primera vegada que, aprenent terminal, `pwd` em va salvar de no saber en quina carpeta estava. En desenvolupament web, `localhost` em va fer un servici paregut: em va situar. La meua aplicació no era encara pública, no tenia domini, no estava desplegada. Estava corrent ací, en este ordinador, en este port, perquè un procés de Vite estava actiu en esta terminal.

I si volia parar-lo:

Ctrl+C

La primera vegada vaig dubtar abans de prémer-lo, com si estiguera tallant un cable roig. Però no. `Ctrl+C` només parava el servidor de desenvolupament. Els arxius seguien allí. El projecte no moria. Simplement deixava de respirar en local.

Hi ha una diferència important entre arxius i processos. Els arxius estan guardats. Un procés està passant. `npm run dev` no crea la teua aplicació del no-res: arranca una eina que llig els arxius, els prepara i els servix al navegador perquè pugues treballar.

Això també explica una confusió habitual: el projecte està bé, però el servidor no és el que tu creus que és. Una vesprada vaig obrir una segona terminal sense recordar que la primera encara tenia el servidor en marxa, i vaig tornar a escriure `npm run dev`. Vite, educadament, em va avisar que el port 5173 estava ocupat i va arrancar en el 5174. Jo, que no llig els avisos quan tinc pressa, vaig continuar mirant la pestanya del 5173, on vivia una versió vella de

l'aplicació. Canviava codi, guardava, i la pantalla no es movia.

Li ho vaig contar a la IA. La resposta va ser un pla d'enginyeria civil:

> Puedes fijar el puerto en vite.config.js con strictPort para evitar que cambie, configurar server.host para exponer el servidor en la red local, y añadir un script "predev" que mate cualquier proceso que esté ocupando el puerto antes de arrancar:
>
> "predev": "kill -9 $(lsof -ti:5173) || true"

Configuració nova, port fixat, un comando amb `kill -9` que jo no sabia llegir però que sonava a arma de foc. Ho vaig apegar tot. Resultat: ara el servidor directament no arrancava, perquè `strictPort` convertia l'avís educat en un error fatal, i el port continuava ocupat per la terminal que jo havia oblidat.

La solució de veritat tenia una tecla i mitja: anar a la primera terminal i prémer `Ctrl+C`. El problema no era el port. Era que jo tenia dos servidors encesos i només en volia un. Vaig desfer el `vite.config.js`,

vaig esborrar l'script del `kill`, i em vaig quedar amb una lliçó reutilitzable: quan la IA proposa configuració per a resoldre un problema, pregunta primer si el problema no serà una cosa oberta que hauries de tancar.

La IA pot ajudar, però cal donar-li informació precisa:

> He ejecutado npm run dev. El terminal muestra localhost:5173. Al abrirlo veo una pantalla en blanco. La consola del navegador muestra este error: ...

Això és molt millor que:

> No funciona.

"No funciona" és una emoció, no un diagnòstic.

Aquell dia vaig començar a tractar el servidor local com una llum de taller: cal saber si està encesa, des d'on s'alimenta i no confondre el fet de veure clar amb haver acabat la faena.

Abans de tancar, vaig escriure en el README:

```
## Desenvolupament
```

Instal·lar dependències:

“‘bash npm install “‘

Arrancar servidor local:

“‘bash npm run dev “‘

Obrir la URL que mostra el terminal. Per parar el servidor: Ctrl+C.

Sí, era bàsic.

Precisament per això valia la pena escriure-ho.

El teu jo futur sempre arriba més cansat del que imagines.

I tot això, de moment, passava dins de casa. El dia que vaig intentar empaquetar l'aplicació per a traure-la fora, la conversa va canviar de to.

Apunt tècnic

`npm run dev` executa un script definit en `package.json`.

On està definit

En `package.json`:

```
{
```

```
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  }
}
```

Quan escrius:

```
npm run dev
```

npm busca el script `dev` i executa el comando associat. En este exemple, `vite`.

Què és `localhost`

`localhost` significa “este mateix ordinador”. És una adreça especial per a accedir a servicis que corren en la teua màquina.

http://localhost:5173/

El número final és el port. Pots imaginar-lo com una porta concreta dins de l’ordinador. Una aplicació pot usar el port 5173, una altra el 3000, una altra el 8000.

El servidor local no és internet

Si una aplicació funciona en `localhost`, això vol dir que funciona en el teu entorn de desenvolupament. No significa que altres persones puguen entrar des de fora.

Per a compartir-la públicament cal desplegar-la en un servici de hosting o en un servidor.

Parar el servidor

En la terminal on està corrent:

```
Ctrl+C
```

Si el servidor es queda penjat o el port ocupat, de vegades cal tancar la terminal o buscar el procés, però en la majoria de casos `Ctrl+C` és suficient.

Errors útils

Quan demanes ajuda a una IA, copia informació concreta:

- comando executat;
- URL que mostra el terminal;
- error complet del terminal;
- error de la consola del navegador;

- què esperaves veure;
- què veus realment.

Prompt útil:

> He arrancado el proyecto con npm run dev. El terminal muestra esta salida:
>
> [pega aquí la salida]
>
> En el navegador pasa esto:
>
> [describe o pega el error]
>
> Ayúdame a diagnosticarlo paso a paso. No propongas cambios todavía. Primero explica las causas probables.

Quan la IA diagnostica abans d'editar, tu aprens més i el projecte sol patir menys.

La pressa vol solucions. L'aprenentatge vol causes. I quasi sempre és ací, en eixa pausa abans del pegat, on deixes de ser usuari d'una resposta i tornes a ser autor del procés.

`Build failed` — quatre hores per una icona

El primer `Build failed` seriós no va arribar quan jo esperava.

En desenvolupament, la meua aplicació de recursos d'aula funcionava. `npm run dev`, navegador, formulari, filtres, targetes. Tot respirava. Jo ja començava a fer eixe gest tan perillós de recolzar-me en la cadira com si dominara alguna cosa.

La IA em va suggerir:

> Antes de publicar, ejecuta npm run build para comprobar que la aplicación se puede generar para producción.

Em va semblar una formalitat. Una ITV abans de traure el cotxe del garatge.

```
npm run build
```

El terminal va tardar menys de dos segons a llevar-me la tonteria.

> src/components/ResourceCard.jsx:3:10
> Module '"lucide-react"' has no exported member 'BookOpenCheck'.
>
> Build failed.

En local no havia fallat. O jo no ho havia vist. L'aplicació arrancava, la icona apareixia o semblava aparéixer, i jo havia assumit que si el navegador no cridava, tot anava bé.

El build és menys diplomàtic.

Vaig copiar l'error a la IA. La resposta va ser ràpida:

> Parece que tu versión de lucide-react no incluye ese icono. Puedes actualizar la dependencia o usar otro icono disponible.

Actualitzar dependència.

Usar una altra icona.

Dues opcions. Una temptadora i gran; una avorrida i menuda. Naturalment, vaig triar la gran.

```
npm install lucide-react@latest
```

El projecte va actualitzar paquets. El build va passar aquell error i en va mostrar un altre. Ara fallava un import d'un component que la IA havia mogut de carpeta en una refactorització anterior. Després un tipus de dada que no coincidia. Després una variable que només existia en desenvolupament. Cada correcció destapava una capa.

Quatre hores després, tenia la sensació d'haver arreglat una persiana i haver descobert que l'edifici no tenia escala.

El pitjor no era el temps. Era la confusió. Jo havia deixat que la IA atacara cada error com si fora independent, però el problema real era de context: el projecte havia anat acumulant decisions menudes sense una comprovació de producció. En local tot semblava prou bé. En build, les excuses s'acabaven.

Vaig parar. Vaig tornar a l'últim commit bo.

Vaig obrir el panell de Source Control en VS Code i vaig mirar els canvis acumulats. La llista era llarga. Vaig fer clic en alguns arxius per a veure el diff visual i vaig entendre que ja no estava depurant, estava negociant amb una madeixa. Vaig fer una

còpia de l'error en un arxiu `notes/build-failed.md` i vaig escriure tres preguntes:

> # Build failed
>
> - Quin és el primer error real? - L'error ve d'una dependència, d'un import o del meu codi? - Quina és la correcció mínima que no canvia res més?

Després vaig tornar a preguntar a la IA, però d'una altra manera:

> No edites todavía. Lee este error de build y dime cuál es la causa mínima. Propón una corrección que toque el menor número de archivos posible. Si hay varias opciones, compáralas.

La resposta va ser molt més útil. El primer error era simplement una icona inexistent en la versió instal·lada. No calia actualitzar mig ecosistema. Podia canviar `BookOpenCheck` per `BookOpen`, que ja estava disponible. Un import. Una línia.

Ho vaig fer.

```
npm run build
```

Un altre error. Però ara era el següent error real, no una conseqüència d'haver sacsejat tot el projecte. El vaig tractar igual: causa mínima, canvi mínim, build.

Eixa vesprada vaig aprendre que un build fallit no és una catàstrofe. És una conversa amb menys adorns que el navegador. Et diu: "açò que pareixia funcionar no es pot empaquetar així".

També vaig aprendre que la IA és molt bona arreglant errors quan li dones un marc estret. Si li dius "arregla el build", pot obrir massa portes. Si li dius "explica el primer error i proposa el canvi mínim", de sobte treballa com algú que respecta una taula d'operacions.

El build va acabar passant.

```
built in 1.43s
```

No vaig celebrar-ho massa. Vaig fer commit. Vaig obrir el panell de Source Control, vaig escriure:

```
Corregeix errors de build
```

I vaig prémer la marca de verda.

I vaig afegir una norma al README:

Abans de publicar, executar sempre:

“‘bash npm run build

```
No perquè el build siga infal·lible. Sinó perquè és una
  manera barata de descobrir que la confiança local era
  només això: local.

Aquella sessió de quatre hores, per cert, va deixar una
  altra víctima que jo encara no veia: la conversa amb
  la IA s'havia fet tan llarga que començava a recordar
  coses que ja no eren certes.

## Apunt tècnic {#apunt-build-failed}

`npm run build` genera la versió de producció de
  l'aplicació. Pot detectar errors que no havies vist
  en desenvolupament.

### Desenvolupament no és producció

`npm run dev` prioritza rapidesa i comoditat. `npm run
  build` comprova si el projecte es pot empaquetar per
  a publicar-lo.

Executa'l sovint:

```bash
npm run build
```
```

Com llegir un error de build

Busca:

- arxiu afectat;
- línia aproximada;
- primer error, no l'últim;
- si parla d'import, dependència, sintaxi, tipus o variable d'entorn.

Prompt útil

> Analiza este error de build. No edites todavía. Dime: 1. cuál es el primer error real; 2. cuál es la causa probable; 3. qué cambio mínimo propones; 4. qué archivos hay que tocar.

La paraula important és "mínim". Quan una aplicació falla, no sempre necessites una reforma. A vegades només necessites canviar una icona.

Un error és informació amb mala cara. Si no el tapes massa prompte amb una solució gran, pot ensenyar-te exactament quina part del teu sistema encara no entens.

`/clear` — l'arxiu que ja no existia

La IA també es cansa, però ho dissimula millor que nosaltres.

No sua. No sospira. No diu "hui no estic fi". Simplement comença a respondre a una versió del projecte que ja no existix.

Em va passar després del build fallit. Portava una conversa llarga: primer havíem creat components, després havíem corregit imports, després havíem discutit si usar `localStorage`, després havíem canviat estils, després tornàrem al build. En algun moment, sense avisar, la IA començà a proposar canvis en un arxiu que jo ja havia eliminat.

> Modifica src/components/Filters.jsx...

Però `Filters.jsx` ja no existia. Ara els filtres vivien dins de `ResourceToolbar.jsx`.

Vaig sentir una irritació injusta. Com si la IA m'haguera fallat personalment. Però la conversa

era un traster. Jo havia anat llançant decisions, errors, correccions i penediments durant hores. Esperar que d'allí isquera una comprensió neta era optimista.

Vaig escriure:

> Estás usando una estructura antigua del proyecto.

La IA va demanar disculpes amb educació automàtica i va continuar quasi igual.

Ahí vaig entendre que no necessitava una disculpa. Necessitava una conversa nova.

Vaig fer dues coses:

Primer, vaig demanar un resum de l'estat actual.

> Resume el estado actual del proyecto en 15 líneas: - objetivo de la app; - estructura de carpetas actual; - decisiones tomadas; - problemas pendientes; - comandos para verificar.

Després vaig copiar el resum, el vaig corregir a mà i el vaig guardar en `prompt.md`.

Només aleshores vaig netejar la conversa.

En algunes eines és `/clear`. En altres és començar un xat nou. En altres és obrir una sessió nova d'agent. El nom canvia; el gest és el mateix: deixar de carregar una motxilla plena de contradiccions.

La conversa nova va ser molt més precisa. Li vaig donar el `prompt.md`, li vaig dir que revisara els arxius actuals i li vaig prohibir assumir estructures anteriors.

> Usa este resumen como punto de partida. Si algo no coincide con los archivos actuales, pregunta antes de proponer cambios.

El canvi va ser immediat. No perquè el model fora més intel·ligent, sinó perquè jo havia deixat de demanar-li que endevinara entre runes.

Netejar context fa por al principi. Sents que perds història. Però la història no és el mateix que el soroll. Si una decisió és important, ha d'estar en un arxiu del projecte, no soterrada en el missatge 138 d'un xat.

Des d'eixe dia vaig començar a distingir tres llocs:

La conversa servix per a pensar.

El codi servix per a executar.

La documentació servix per a recordar.

Quan una conversa es fa massa llarga, intente salvar allò que mereix memòria i deixar morir la resta. És una forma de higiene. Igual que no guardes tots els esborranys d'una pissarra, tampoc cal arrossegar totes les respostes de la IA fins al final del projecte.

La IA no necessita saber-ho tot.

Necessita saber què és cert ara.

I si cada conversa nova necessitava que jo li contara què era cert, potser eixe relat mereixia deixar de ser improvisat i convertir-se en un arxiu del projecte.

Apunt tècnic

Netejar el context és útil quan la IA:

- cita arxius que ja no existixen;
- proposa solucions ja descartades;
- barreja decisions antigues i noves;

- empitjora el mateix bug després de tres o quatre intents;
- respon amb massa seguretat sobre una estructura que ha canviat.

Abans de netejar

Guarda un resum útil:

> # Estat del projecte
>
> - Objectiu: - Stack: - Com arrancar: - Estructura actual: - Decisions importants: - Problemes pendents: - Últim commit bo:

Després de netejar

Dona context curt i verificable:

> Este es el estado actual del proyecto. Lee primero los archivos antes de proponer cambios. No asumas decisiones de conversaciones anteriores.

Netejar la conversa no és perdre memòria. És decidir quina memòria mereix passar al projecte.

`prompt.md` — el dia que vaig deixar de demanar favors

Durant molt de temps vaig tractar els prompts com si foren desitjos. Ni tan sols la higiene del `/clear` m'havia curat d'això: netejava la conversa i tornava a començar-la amb la mateixa vaguetat de sempre.

> Hazlo más bonito. Arregla esto. Añade login. Mejora el código.

Són frases humanes, i per això funcionen a mitges. Una persona que et coneix pot entendre el context, llegir-te la cara, recordar la conversa de la setmana passada i imaginar què vols dir amb "més bonic". Una IA no té eixe luxe. Té text, arxius, instruccions i probabilitats.

L'última gota va caure en una sessió acabada d'estrenar. Volia que el formulari d'afegir recursos

fora una mica més còmode —el cursor no anava al primer camp, res greu— i vaig escriure, amb tota la mandra del món:

> Mejora el formulario de añadir recursos
> para que sea más cómodo de usar.

L'agent s'ho va prendre com un xec en blanc. Quan va acabar, `git status` llistava nou arxius modificats. Havia partit el formulari en tres components, afegit una llibreria de validació i una altra de notificacions —dues dependències noves que jo no havia autoritzat—, i cada camp tenia ara missatges d'error animats, en castellà, per a una aplicació en valencià. Tot molt professional. Tot per a un problema que es resolia amb un atribut `autofocus`.

El pitjor és que funcionava. Si no haguera mirat l'estat del projecte, ho hauria acceptat. Vaig tardar més a desfer-ho —`git restore`, esborrar les dependències, comprovar que tot tornava a arrancar— del que hauria tardat a escriure l'atribut a mà.

La culpa no era de l'agent. Jo havia demanat "més còmode" i ell havia omplit el buit amb ambició. Un

prompt vague no és una petició menuda: és una invitació a sobredimensionar.

I hi havia un detall que em va coure més que les dependències: aquells missatges d'error en castellà. L'agent no ho havia decidit per caprici. Jo li parlava en castellà des del primer dia, per inèrcia, com fa quasi tothom quan comença copiant prompts d'internet. Li estava demanant una aplicació en valencià amb ordes en una altra llengua, i encara m'estranyava que es fera un embolic. No sols demanava malament: ni tan sols demanava en la meua llengua. Aquell dia vaig decidir dues coses. La primera, escriure encàrrecs en lloc de favors. La segona, escriure'ls en valencià: si volia que la ferramenta parlara com la meua aula, el primer que havia de parlar com la meua aula era jo.

El dia que vaig crear `prompt.md` va ser el dia que vaig deixar de demanar favors i vaig començar a escriure encàrrecs.

El projecte de recursos d'aula ja tenia prou història: un prototip, una versió mínima, components, Git, build, alguns errors i una decisió pendent sobre persistència. També tenia calendari: era mitjan

maig, quedaven cinc setmanes de curs, i jo volia arribar al setembre amb una ferramenta, no amb una promesa. Cada vegada que obria una sessió nova amb la IA, havia d'explicar massa coses. I com més explicava de memòria, més variava la versió.

Vaig crear un arxiu:

```
nano prompt.md
```

I vaig escriure:

> # Context del projecte
>
> Aplicació web per a guardar recursos educatius. Stack actual: Vite + React. Ara mateix les dades viuen en memòria; encara no hi ha persistència.
>
> # Normes
>
> - No afegir dependències sense justificar-ho. - No canviar l'estructura de carpetes sense preguntar. - Fer canvis menuts. - Explicar quins arxius es tocaran abans d'editar.
>
> # Verificació
>
> Després de cada canvi important:

“‘bash npm run build

```
Era un document modest, però canvià el to de les
  converses. Ja no començava amb una súplica:

"A veure si m'entens."

Començava amb un contracte:

"Este és el terreny."

La IA respon millor quan li dones límits. Això pot
  semblar contradictori, perquè tendim a pensar que la
  gràcia d'una IA és que faça de tot. Però "de tot" és
  una mala especificació. En software, la llibertat
  absoluta sol produir soroll absolut.

Vaig provar-ho amb una tasca concreta:

```text
Usa prompt.md com a context.
Vull afegir localStorage perquè els recursos no es
  perden en recarregar.
Abans d'editar, proposa un pla en tres passos i
  digues-me quins arxius tocaràs.
```

La resposta ja no va ser una allau de codi. Va proposar:

1. Llegir recursos inicials des de `localStorage`.
```

2. Guardar cada canvi quan s'afegira o eliminara un recurs.
3. Afegir una funció auxiliar per no repetir codi.

Arxius: només `src/App.jsx`.

Això era revisable. Podia dir sí, no o "fes només el primer pas". El prompt havia convertit una nebulosa en una tasca.

També vaig descobrir que un bon prompt no ha de ser llarg. Ha de ser concret. Hi ha prompts de mitja pàgina que no diuen res i tres línies que estalvien una hora.

La fórmula que més m'ha servit és esta:

> Objectiu: Context: Restriccions: Criteri d'acceptació: Verificació:

Quan falta l'objectiu, la IA optimitza qualsevol cosa.

Quan falta el context, inventa.

Quan falten restriccions, toca massa.

Quan falta criteri d'acceptació, no sap quan ha acabat.

Quan falta verificació, tot sembla correcte fins que algú ho prova.

Un prompt no és una vareta. És una orde de treball.

I una orde de treball bona no lleva creativitat. La concentra.

El pas següent era donar eixes ordes de treball a una ferramenta que no sols opinara sobre els meus arxius, sinó que els poguera tocar.

Apunt tècnic

Un `prompt.md` pot viure en l'arrel del projecte i servir com a context estable per a sessions amb IA.

Plantilla mínima

```
# Projecte

Descripció curta de l'aplicació.

# Stack

Tecnologies principals.

# Estat actual

Què funciona i què no.

# Normes per a la IA
```

> - No afegir dependències sense preguntar. - No fer refactors no demanats. - Tocar el mínim d'arxius possible. - Explicar el pla abans d'editar.
>
> # Verificació
>
> “‘bash npm run build “‘

Prompt d’encàrrec

> Usa prompt.md com a context. Objectiu: [què vull aconseguir] Restriccions: [què no vull tocar] Criteri d'acceptació: [com sabrem que està fet] Verificació: [quins comandos cal executar] Abans d'editar, proposa un pla curt.

La IA no necessita un poema. Necessita condicions de treball.

Un bon prompt no és una frase enginyosa: és una manera d’assumir responsabilitat abans que el codi existisca. Quan escrius límits, també escrius qui vols ser dins del projecte.

`opencode` — l'agent que parlava anglés en el pitjor moment

La diferència entre un xat i un agent es nota quan deixa de respondre amb fragments i comença a mirar els teus arxius. Ara jo tenia un `prompt.md` amb les normes escrites; tocava comprovar si un agent de veritat les respectava.

Amb OpenCode vaig tindre eixa sensació per primera vegada d'una manera clara. No era només "copia este codi i enganxa'l". Era més incòmode i més potent: l'eina podia inspeccionar el projecte, proposar canvis i editar fitxers.

La primera temptació va ser deixar-la fer.

La segona, més saludable, va ser posar-li normes.

Vaig obrir el projecte de recursos d'aula i vaig escriure una tasca menuda:

> Revisa el projecte i proposa com afegir una confirmació abans d'esborrar un recurs. No edites encara.

Eixa última frase és una tanca. Menuda, però important.

OpenCode va llegir l'estructura, va identificar on es cridava la funció d'esborrar i va proposar un canvi limitat. No era màgia. Era una lectura ràpida de coses que jo també hauria pogut llegir, però amb una paciència que a mi se m'acabava abans.

Després li vaig deixar editar.

> Aplica el canvi. Toca només App.jsx.

El fet que jo escriguera "toca només" em feia sentir una mica autoritari. Després vaig comprovar que era higiene. Un agent sense límits pot decidir que, ja que està, també reorganitza estils, canvia noms, mou components o "millora" textos. No per maldat. Perquè optimitza una idea general de qualitat, no necessàriament la meua necessitat concreta.

OpenCode va fer el canvi. Vaig mirar el diff. Vaig executar:

```
npm run build
```

Passava.

Vaig provar en el navegador. En prémer esborrar, apareixia un avís, i el recurs només desapareixia si confirmava.

La primera versió, però, tenia un problema de to. El missatge de confirmació deia:

> Are you sure you want to delete this item?

No era greu. També era una pista. L'agent havia resolt la funció, però havia introduït una veu que no pertanyia a l'aplicació. Jo estava escrivint una ferramenta en valencià per a ús propi i docent; de sobte, en el moment de major risc —esborrar—, l'aplicació parlava en anglés genèric.

Li vaig demanar el canvi:

> Canvia només el text de confirmació a valencià. No toques la lògica.

Va respondre amb:

> Segur que vols eliminar este recurs?

Millor.

Després vaig provar una altra cosa: vaig prémer Cancel·lar. El recurs continuava. Vaig prémer Acceptar. Desapareixia. Vaig provar amb una llista filtrada. També. Fins aquell moment jo havia provat el camí que volia que funcionara; ara començava a provar també el camí que havia de protegir-me.

Era un canvi menut, però per a mi va ser una frontera. Fins eixe moment, la IA m'havia ajudat a escriure codi. Ara m'ajudava a modificar un projecte real sense que jo deixara de dirigir-lo.

També em va agradar una altra cosa: OpenCode no amagava tant el fet que estava treballant en fitxers. Em recordava que allò no era una conversa etèria, sinó una operació sobre una carpeta del meu ordinador. Eixa sensació de propietat m'importa. Les ferramentes obertes o més transparents no són bones automàticament, però et conviden a mirar el mecanisme.

La lliçó no va ser "usa OpenCode". Les eines canviaran. La lliçó va ser: quan un agent pot editar, la teua faena canvia. Ja no consistix només a enten-

dre codi. Consistix a definir encàrrecs, posar límits, revisar canvis i provar resultats.

Un agent no és un becari, encara que de vegades el tractem així. Tampoc és un mestre. És més estrany: una capacitat de treball sense criteri propi sobre el teu context humà.

Pot fer molt.

Per això mateix cal dir-li menys.

Menys, però millor.

Amb la primera eina domesticada, vaig voler provar-ne una altra amb un encàrrec més delicat. I vaig aprendre una paraula nova pel camí.

Apunt tècnic

Quan uses un agent de codi, separa tres fases.

1. Inspecció

> Revisa el projecte i explica on faries el canvi. No edites encara.

2. Pla

> Proposa un pla curt. Indica quins arxius tocaràs i per què.

3. Edició i verificació

Aplica el canvi mínim. Després indica com verificar-lo.

Regles útils

- Fer commit abans de canvis grans.
- Demanar que no afija dependències sense permís.
- Limitar arxius quan siga possible.
- Revisar `git diff`.
- Executar proves o build.
- Provar Acceptar i Cancel·lar en accions destructives.
- Revisar textos introduïts per la IA: idioma, to i coherència.

Encàrrecs millors

Pitjor:

Millora l'esborrat.

Millor:

Afig confirmació abans d'esborrar un recurs. Toca només App.jsx. El text ha

> d'estar en valencià. No afegis dependències. Després indica com provar Acceptar i Cancel·lar.

L'agent pot escriure molt de pressa. La teua responsabilitat és reduir la superfície de l'error.

La potència sense marc és només soroll amb bons modals. Un agent et fa més capaç quan t'obliga a ser més precís, no quan et permet desaparéixer.

`codex` — cometes, comes i mala idea

Amb Codex vaig aprendre una paraula que no apareix en cap tutorial: abdicar.

Delegar està bé. Abdicar, no.

La diferència és subtil fins que tens un agent treballant en el teu projecte. Delegar és dir: “fes esta tasca amb estes condicions”. Abdicar és dir: “fes que vaja bé” i apartar-te de la pantalla com si el resultat ja no tinguera a veure amb tu.

Vaig provar Codex amb una tasca concreta: afegir un botó per exportar els recursos a CSV.

Abans, jo hauria escrit:

> Añade exportación a CSV.

Esta vegada vaig escriure:

> Objectiu: afegir exportació CSV dels recursos. Restriccions: no afegir dependències; no canviar l'estructura visual. Criteri: ha d'aparéixer un botó ”Exportar

> CSV” i descarregar un fitxer. Verificació: npm run build i prova manual. Abans d'editar, proposa pla.

Codex va inspeccionar el projecte i va proposar un pla raonable: funció per convertir recursos a CSV, crear un `Blob`, generar un enllaç temporal, descarregar. Tot en el component principal o en un helper menut.

Vaig acceptar.

El canvi va ser ràpid. Massa ràpid per al meu ego, però no per al meu criteri. Vaig mirar el diff. Hi havia una funció que escapava cometes, separava camps amb comes i posava una capçalera.

Vaig provar. El CSV es descarregava.

Després vaig obrir-lo. Les paraules amb accents es veien bé. Les comes dins d'un títol també. Vaig afegir un recurs amb cometes per fer la guitza. També funcionava.

El detall de les cometes em va donar una alegria absurda. Havia escrit:

> Activitat ”La ciutat invisible”

En altres temps hauria pensat que això era una mania meua. Però els CSV són precisament el territori on les manies es convertixen en bugs: comes, salts de línia, accents, fulls de càlcul que interpreten coses, programes que separen per punt i coma segons la configuració regional. El fitxer es descarregava, sí. Però un CSV que només funciona amb títols perfectes no és una exportació; és una demostració.

Vaig demanar una revisió específica:

> Revisa la funció d'exportació CSV. Comprova casos amb comes, cometes, accents i salts de línia. No canvies la interfície.

Codex va ajustar l'escapat i afegí un comentari curt. Jo vaig crear tres recursos de prova amb mala idea:

> Matemàtiques, problemes Lectura "guiada" Ciències amb salt de línia

L'últim era quasi una provocació. Però millor provocar jo en local que descobrir-ho quan algú exportara una llista real.

També vaig comprovar el nom de l'arxiu. La primera versió descarregava:

> resources.csv

Vaig canviar-lo per:

> recursos-aula.csv

No és només traducció. És coherència. Un projecte xicotet també té una veu.

Per primera vegada vaig sentir que un agent no m'havia substituït, m'havia ampliat. Jo probablement hauria buscat "download csv javascript" i hauria copiat una solució de Stack Overflow amb la mateixa esperança i menys traçabilitat. Codex ho havia fet dins del projecte, amb els noms reals i el meu context.

Però la part important va vindre al final, quan em va resumir:

> Canvis: - Afegida funció exportResourcesToCsv. - Afegit botó d'exportació. - Verificat amb npm run build.

Eixe resum no era literatura. Era el primer esborrany del meu commit.

```
git diff
git status
git add .
git commit -m "Afig exportacio CSV"
```

La seqüència em va semblar madura: encàrrec, pla, canvi, verificació, diff, commit. No perquè siga burocràtica, sinó perquè cada pas respon a una pregunta.

Què vols?

Com ho faràs?

Què has canviat?

Funciona?

Ho vols conservar?

La IA pot accelerar molt el centre del procés. Però les preguntes del voltant continuen sent humanes.

I una d'eixes preguntes —què has canviat exactament?— m'esperava en un lloc que jo encara mirava massa poc.

Apunt tècnic

Un bon flux amb Codex o qualsevol agent similar:

1. Fer commit o comprovar que estàs en un punt net.
2. Escriure una tasca concreta.
3. Demanar pla abans d'editar.
4. Limitar arxius o dependències.
5. Revisar el diff.
6. Executar verificació.
7. Fer commit amb un missatge clar.

Per a exportacions, prova dades molestes:

- comes;
- cometes;
- accents;
- camps buits;
- salts de línia;
- noms d'arxiu comprensibles.

Prompt base:

> Objectiu: Restriccions: Criteri d'acceptació: Verificació:

> Abans d'editar, inspecciona el projecte i proposa un pla curt.

Prompt de casos límit:

> Abans de donar per bona esta funció, proposa casos límit que podrien trencar-la. Després indica com provar-los manualment.

Delegar no és desaparéixer. És crear les condicions perquè una altra intel·ligència treballe sense traure't del lloc de decisió.

`git diff` — el «Crear recurso» que ningú havia demanat

La primera vegada que vaig entendre de veritat la revisió de canvis no va ser llegint un manual. Va ser desconfiant d'una solució que funcionava.

Codex havia afegit l'exportació CSV i tot semblava correcte. El botó apareixia, el fitxer es descarregava, el build passava. La part mandrosa del meu cervell deia: commit i avant.

Però vaig obrir el panell de Source Control en VS Code i vaig fer clic en l'arxiu que havia canviat.

I vaig començar a llegir el diff visual.

Al principi, el diff fa una mica de respecte. Línies verdes amb codi nou, línies rojes amb codi eliminat, fragments de codi sense tot el context. Però després l'ull s'acostuma. El diff és una conversa molt honesta: açò estava, açò ja no està, açò és nou.

La funció CSV estava bé. El botó també. Però entre els canvis hi havia una modificació menuda en un text de la interfície. L'agent havia canviat "Afegir recurs" per "Crear recurso". Ni era greu ni trencava res. Però no ho havia demanat. I, sobretot, canviava de valencià a castellà.

Eixe detall em va ensenyar més que un error gran.

Un agent pot fer un canvi correcte i, alhora, colar una decisió que no li correspon. No per malícia. Per inèrcia. Per context mixt. Perquè en algun prompt jo havia escrit en castellà. Perquè el model completa patrons.

Vaig revertir eixa línia a mà.

Després vaig tornar a mirar el diff. Ara sí: només exportació CSV.

Vaig fer commit.

Des d'aleshores, revisar el diff és el moment en què deixe de mirar el resultat i mire la intervenció. Una app pot funcionar i, malgrat això, haver canviat coses que no tocaven. Pot compilar i haver afegit dependències innecessàries. Pot veure's millor i ha-

ver trencat accessibilitat. Pot resoldre el bug i haver reformat mitja casa.

El diff no és una garantia absoluta. Cal entendre prou per a llegir-lo. Però inclús quan no entens cada línia, pots detectar patrons:

Per què ha tocat este arxiu?

Per què ha canviat textos?

Per què ha afegit una dependència?

Per què hi ha més eliminacions que afegits?

Per què ha modificat configuració si jo només demanava un botó?

Llegir el diff és una forma d'educar la pròpia sospita. No una sospita paranoica, sinó professional: abans d'acceptar un canvi, mira'l.

La IA treballa ràpid.

El diff t'obliga a tornar a la velocitat humana.

Poc després, un botó de la meua aplicació va deixar de fer res. I l'origen d'aquell silenci, encara que jo no ho sabia, estava en un diff que no havia llegit.

Apunt tècnic

Amb VS Code

El panell de Source Control de VS Code permet revisar canvis de manera visual. Cadascun dels arxius modificats es pot obrir fent clic ahí.

El diff visual Quan feu clic en un arxiu de la llista "Canvis", VS Code mostra dues columnes: a l'esquerra l'estat anterior, a la dreta l'estat actual. Les línies noves apareixen en verd, les eliminades en roig. Això permet detectar fàcilment què ha canviat sense llegir el diff en mode text.

Què mirar

- Arxius tocats que no esperaves.
- Textos canviats sense motiu.
- Dependències noves en `package.json`.
- Canvis de configuració.
- Refactors barrejats amb un bugfix.
- Eliminacions grans.

Descartar canvis visuals Si un arxiu té canvis que no voleu conservar, podeu fer clic en el ~ al costat del nom per a tornar-lo a l'estat anterior abans de fer commit.

Amb el terminal

Si preferiu el terminal, estos són els comandaments equivalents.

Veure l'estat dels canvis:

```
git status
```

Veure el diff de tot el projecte:

```
git diff
```

Veure el diff d'un arxiu concret:

```
git diff src/App.jsx
```

Veure canvis ja preparats amb `git add`:

```
git diff --staged
```

Prompt útil

> Revisa este diff com si fora una revisió de codi. Digues si hi ha canvis no relacionats amb la tasca, riscos o coses que convé separar en un altre commit.

No cal entendre-ho tot per començar a revisar. Cal començar a mirar.

Mirar el diff és un gest menut, però canvia la postura interior: deixes de rebre el futur del projecte com un regal i comences a acceptar-lo com una decisió.

`console.log()` — el botó que callava

Hi ha errors que tenen la decència de cridar.

Este no.

Jo premia “Guardar recurs” i la pantalla no canviava. Cap missatge roig, cap explosió, cap stack trace. El formulari es quedava allí, indiferent, com un funcionari a cinc minuts d’esmorzar.

Vaig preguntar a la IA:

> El botó no funciona.

Mala pregunta.

La IA va contestar amb una llista de possibilitats: event listener, estat, formulari, validació, render. Tot podia ser. I quan tot pot ser, no tens diagnòstic; tens meteorologia.

El pitjor d’aquella resposta és que semblava útil. Tenia paraules tècniques, enumerava causes plausibles i acabava proposant tres canvis. Si haguera tingut pressa —i en tenia— hauria pogut aplicar el

primer pegat, després el segon, després el tercer. Hauria acabat amb el mateix botó trencat i tres modificacions més per entendre.

Vaig parar. No per saviesa, sinó per cansament. Hi ha un moment en què fins i tot la desesperació necessita mètode.

Vaig recordar una cosa simple:

```
console.log("submit");
```

El vaig posar al principi de la funció que guardava el recurs. Obrí la consola del navegador, premí el botó i no aparegué res.

Això ja era informació. La funció no s'estava executant.

Vaig posar un altre log en el component. Després un en el formulari. Després vaig mirar l'HTML renderitzat amb l'inspector. El botó tenia `type="button"` en lloc de `type="submit"`. La IA, en una refactorització anterior, havia canviat una cosa aparentment innocent i havia tallat el camí de l'esdeveniment.

El vaig corregir:

```
<button type="submit">Guardar recurs</button>
```

Ara el log apareixia.

Després el recurs no es veia. Un altre `console.log`.

```
console.log("recursos actuals", recursos);
```

El recurs entrava en l'array, però la llista filtrada usava una variable antiga. Un segon error, amagat darrere del primer.

Vaig fer una cosa que m'hauria semblat ridícula una setmana abans: dibuixar el camí de la dada en una llibreta.

> formulari -> guardarRecurs -> recursos ->
> filtre -> llista visible

I vaig posar un log en cada porta.

```
console.log("1 formulari", { titol, url });
console.log("2 recurs creat", recurs);
console.log("3 recursos", recursosActualitzats);
console.log("4 filtre", filtre);
console.log("5 visibles", recursosFiltrats);
```

No era elegant. Era gairebé infantil. Però en aquell mapa infantil el bug deixà de ser una boira i es convertí en una frontera concreta: fins ací arriba bé, a partir d'ací es perd.

També vaig descobrir una cosa que la IA no podia saber si jo no li ho deia: l'error només apareixia quan hi havia un filtre escrit. Amb el buscador buit, el recurs nou sí que es veia. Amb "mates" escrit, no. El problema no era guardar; era recalcular la vista després de guardar.

La pregunta següent ja no va ser:

> El botó no funciona.

Va ser:

> En guardar un recurs, l'array s'actualitza correctament. Ho veig en este log:
>
> [enganxa log]
>
> Però si hi ha un filtre actiu, la llista visible no inclou el recurs nou fins que esborre i torne a escriure el filtre. Ajuda'm a localitzar on es recalcula recursosFiltrats. No proposes encara una refactorització.

La resposta va canviar de qualitat. Ja no era meteorologia. Era diagnòstic.

Eixa vesprada vaig entendre que depurar és fer visible allò que el programa ja sap però tu encara

no. La IA pot ajudar molt, però si només li dius “no funciona”, li demanes endevinació. Si li dones logs, captures i passos, li dones realitat.

El `console.log` és poc elegant. També ho és encendre la llum per buscar les claus. No sempre cal una ferramenta sofisticada; de vegades cal veure on cau la dada.

Després de corregir el bug, vaig llevar els logs sorollosos. No tots. Alguns logs temporals són bastides: ajuden a construir, però no han de quedar en la façana.

En el commit vaig escriure:

```
git commit -m "Corregeix guardat amb filtre actiu"
```

El missatge era menys important que el que havia passat abans: per primera vegada no havia acceptat una solució de la IA com qui accepta un remei. Havia portat el problema fins a un lloc on la solució tenia sentit.

Però el bug m’havia costat una vesprada, i jo no volia pagar eixa vesprada cada vegada. Necessitava algú que fera la ronda de vigilància per mi.

Apunt tècnic

Per depurar una acció:

1. Posa un log al principi de la funció.
2. Comprova si s'executa.
3. Mostra les dades d'entrada.
4. Mostra l'estat després del canvi.
5. Revisa la consola del navegador.
6. Reduïx el cas: què has de fer exactament perquè falle?
7. Lleva els logs temporals abans de tancar el canvi.

Exemple:

```
function guardarRecurs(event) {
  event.preventDefault();
  console.log("submit rebut");
  console.log("títol", titol);
}
```

Per seguir el camí d'una dada:

```
console.log("entrada", dadesDelFormulari);
console.log("transformació", recursCreat);
console.log("estat nou", recursosActualitzats);
console.log("vista", recursosFiltrats);
```

Què donar-li a la IA

Millor que “no funciona”:

- passos exactes per reproduir l’error;
- què esperaves;
- què passa realment;
- logs rellevants;
- captura o text de la consola;
- últim canvi que recordes haver fet.

Prompt útil:

> No em dones encara una solució. Indica quins console.log afegiries i on per saber en quin punt es trenca el flux.

Depurar no és provar solucions a cegues. És reduir la foscor.

Hi ha una ètica de la llum en això: abans d’opinar, mirar; abans de corregir, entendre; abans de demanar-li a una IA que actue, fer visible el que està passant.

`npm test` — dotze verds que no vigilaven res

Després del bug del botó vaig fer inventari de com verificava la meua aplicació.

El mètode era este: guardar l'arxiu, esperar que el navegador es recarregara, escriure "matemàtiques" en el buscador, mirar si eixien els recursos de matemàtiques, afegir un recurs de prova, mirar si apareixia, esborrar-lo, i respirar.

Cada canvi pagava el mateix peatge. I com que el peatge era car, de vegades no el pagava. Canviava una cosa xicoteta, em semblava innocent, i ho donava per bo sense mirar. Així era exactament com m'havia colat el `type="button"`.

Li ho vaig comentar a la IA, més com a queixa que com a pregunta. Em va contestar amb una paraula que jo associava a la gent seriosa: tests.

> Podem afegir tests automàtics amb Vitest. Cada vegada que canvies alguna co-

> sa, executes npm run test i comproves que res s'ha trencat.

La promesa era llaminera: que la màquina fera la ronda de vigilància per mi.

Vaig instal·lar Vitest, vaig afegir l'script i vaig demanar tests per a la funció que filtrava recursos. La IA va generar un arxiu sencer en uns segons. Dotze tests. Els vaig executar.

Dotze verds.

A la primera.

Em vaig sentir professional durant un minut i mig. Dotze comprovacions automàtiques vigilant el meu codi. Una xicoteta plantilla de funcionaris incansables.

Després em va picar una sospita. Dotze tests escrits en trenta segons per a un codi que la mateixa IA havia escrit, i cap fallava? Ni un? Jo, que portava setmanes trobant errors a pales, havia produït per fi una cosa perfecta?

Vaig fer una prova que no venia en cap manual: vaig sabotejar el meu propi codi. Vaig obrir la funció de filtrar i vaig comentar la línia que filtrava. Ara la

funció tornava sempre tots els recursos, buscares el que buscares. Un error de manual.

Vaig executar els tests.

Dotze verds.

La meua plantilla de funcionaris incansables no vigilava res. Feien la ronda amb els ulls tancats.

Vaig llegir els tests d'un en un, esta vegada de veritat. N'hi havia que comprovaven coses com que el resultat de filtrar era un array. Ho era sempre, encara que el filtre estiguera trencat. Uns altres feien una cosa més fina i més perversa: calculaven el resultat esperat cridant la mateixa funció que estaven provant. La funció, comparada amb ella mateixa, coincidia sempre. Com un examen on l'alumne redacta les preguntes mirant les seues pròpies respostes.

Cap test contenia una expectativa meua. Tots contenien expectatives del codi sobre el codi.

Vaig entendre aleshores una cosa que no m'havia dit ningú: un test no val pel verd. Val pel roig que és capaç de donar. Un test que no pot fallar no és una comprovació; és decoració.

Vaig canviar el mètode. Abans de demanar tests, escrivia jo, en llenguatge normal, què hauria de trencar-los:

> Escriu tests per a la funció de filtrar recursos. Comprova comportament, no implementació: - buscar ”mates” troba ”Matemàtiques” (majúscules i accents a banda); - buscar text que no existix torna llista buida; - buscar text buit torna tots els recursos. No uses la funció provada per calcular el resultat esperat: escriu els valors esperats a mà.

Els tests nous eren menys. Cinc en lloc de dotze. Però quan vaig repetir el sabotatge —la línia comentada, el filtre trencat—, tres es van posar rojos immediatament.

El primer test que em va fallar de veritat em va alegrar la vesprada. Sé que sona estrany celebrar un fracàs, però aquell roig volia dir que algú, per fi, estava mirant.

Amb la ronda de vigilància muntada, podia tornar al deute més antic del projecte: les dades que encara es perdien en recarregar la pàgina.

Apunt tècnic

Instal·lar Vitest en un projecte Vite:

```
npm install -D vitest
```

En `package.json`:

```
"scripts": {
  "test": "vitest"
}
```

Executar:

```
npm run test
```

Un bon test

- Comprova **un sol comportament**.
- Té un nom que descriu eixe comportament (“torna llista buida si no hi ha coincidències”).
- Compara amb valors esperats escrits a mà, no calculats amb el codi que es prova.
- **És capaç de fallar.** Si no saps què l’hauria de posar en roig, encara no és un test.

Exemple:

```
import { describe, it, expect } from "vitest";
import { filtrarRecursos } from "./recursos";

describe("filtrarRecursos", () => {
  it("troba recursos ignorant majúscules", () => {
    const recursos = [{ titol: "Matemàtiques 3r" }, {
      titol: "Lectura" }];
    expect(filtrarRecursos(recursos, "mate")).toEqual([
      { titol: "Matemàtiques 3r" },
    ]);
  });

  it("torna llista buida si no hi ha coincidències", ()
    => {
    const recursos = [{ titol: "Lectura" }];
    expect(filtrarRecursos(recursos,
      "física")).toEqual([]);
  });
});
```

La prova del sabotatge

Trenca el codi expressament (comenta una línia clau) i executa els tests. Si continuen verds, no tens tests: tens atrezzo.

Prompt útil:

> Escriu tests per a esta funció. Abans dels tests, llista quins comportaments hauri-en de fer-los fallar. Comprova comporta-ment observable, no detalls interns. No repliques la implementació per calcular els valors esperats: escriu-los a mà.

Un test verd només tranquil·litza si has vist que sap posar-se roig.

`localStorage` — el recurs que va sobreviure a un F5

La primera vegada que un recurs va sobreviure a un `F5`, vaig sentir una emoció desproporcionada. El deute era vell: venia d'aquell mapa de climes que va desaparéixer davant dels meus ulls la primera vesprada del projecte.

Fins aleshores, la meua aplicació tenia memòria de peix. Afegia recursos mentre la pàgina estava oberta, però en recarregar tot desapareixia. Era com parlar amb algú que et mira amb atenció i, en girar-se, oblida el teu nom.

La IA em va proposar `localStorage`.

> Podem guardar els recursos en localStorage perquè persistisquen en el navegador sense muntar encara una base de dades.

La frase tenia música. Persistir sense base de dades. Continuar sense muntar un servidor. Una solució intermèdia.

Vaig demanar el canvi amb prudència:

> Afig localStorage amb el mínim canvi possible. No afegis dependències. Explica on es llig i on es guarda.

El mecanisme era senzill. En arrancar, l'aplicació intentava llegir dades guardades. Quan els recursos canviaven, els escrivia.

```
localStorage.setItem("recursos",
  JSON.stringify(recursos));
```

I per recuperar:

```
JSON.parse(localStorage.getItem("recursos") || "[]");
```

Vaig afegir un recurs, vaig recarregar i continuava allí.

Durant uns segons vaig pensar: ja està, tinc base de dades.

No.

Tenia una llibreta dins del navegador. Útil, ràpida, privada en el sentit domèstic de la paraula, però

limitada. Si obria l'aplicació en un altre ordinador, no hi havia dades. Si netejava el navegador, podien desaparéixer. Si volia compartir recursos amb un company, `localStorage` no servia.

Ho vaig comprovar de la manera menys científica possible. Vaig obrir la mateixa aplicació en Firefox, després en Chromium, després en el mòbil. En cada lloc, la biblioteca començava de zero, com si jo no haguera treballat mai. Durant un minut em vaig indignar amb la tecnologia, que és una forma molt còmoda de no admetre que no havia entés la ferramenta.

`localStorage` no guardava "en la meua aplicació".

Guardava en eixe navegador concret, en eixe dispositiu concret, per a eixe domini concret.

La diferència és menuda només si no tens dades importants.

Vaig imaginar una escena prou real: preparar una llista de recursos el diumenge a la nit en el portàtil de casa, arribar dilluns al centre, obrir la mateixa URL en l'ordinador de l'aula i trobar-la buida. La ferramenta no havia fallat. Havia fet exactament

allò per a què l'havia dissenyada. El que fallava era la meua expectativa.

Però per a moltes ferramentes personals o prototips, era suficient. I això també és una lliçó: no totes les aplicacions necessiten començar amb usuaris, login, servidor i base de dades. A vegades la versió honesta és local, exportable i simple.

Vaig afegir dos botons més: exportar i importar JSON. Això convertia la llibreta del navegador en una cosa que podia copiar, guardar i traslladar.

El primer export va ser un arxiu amb un nom horrorós:

> data.json

La IA l'havia proposat sense malícia. Jo l'hauria acceptat sense mirar. Però després vaig pensar en el meu escriptori, eixe cementeri de documents amb noms com `documento-final-bueno-ahora-si.pdf`, i vaig canviar-lo:

> recursos-aula-2026-05-19.json

Una tonteria, sí. També una forma de respecte pel jo futur.

Després vaig provar la importació amb una còpia exportada. Funcionava. Vaig provar amb un JSON trencat, canviant una clau a mà. L'aplicació es va quedar en blanc. La IA havia implementat el camí feliç: si l'arxiu era correcte, tot bé. No havia pensat en el docent que descarrega, renomena, envia per correu, obri amb un editor i sense voler es carrega una coma.

Li vaig demanar que no reescriguera tot:

> La importació JSON ha de fallar de manera amable. Si l'arxiu no és vàlid, mostra un missatge i no substituïsques les dades actuals. Canvi mínim.

Eixa frase —"no substituïsques les dades actuals"— era la part important. Un error d'importació no havia de convertir-se en una pèrdua de treball.

També vaig afegir un avís menut abans d'importar:

> La importació substituirà els recursos actuals. Exporta una còpia abans si vols conservar-los.

No és una frase bonica. És una frase necessària.

No era perfecte.

Era comprensible.

I, en aquell moment, comprensible era millor que ambiciós.

Però jo ja sabia que la llibreta tenia data de caducitat: la prova del dilluns al matí, en l'ordinador de l'aula, no la passaria.

Apunt tècnic

`localStorage` guarda dades en el navegador com a text.

Guardar:

```
localStorage.setItem("recursos",
  JSON.stringify(recursos));
```

Llegir:

```
const recursos =
  JSON.parse(localStorage.getItem("recursos") || "[]");
```

Eliminar:

```
localStorage.removeItem("recursos");
```

Quan és útil

- Prototips.
- Eines personals.
- Preferències d'usuari.
- Dades no sensibles.
- Aplicacions sense compte d'usuari.

Límits

- No sincronitza entre dispositius.
- No és una base de dades compartida.
- Pot esborrar-se si neteges el navegador.
- No és lloc per a secrets o dades delicades.
- No és una còpia de seguretat.
- La importació ha de validar l'arxiu abans de substituir dades.

Exportar sense atrapar dades

Si uses `localStorage`, pensa en una eixida:

- exportar JSON;
- importar JSON;
- avisar abans de substituir;
- posar noms d'arxiu amb data;

- documentar on viuen les dades.

Prompt útil per a validar importació:

> Revisa esta funció d'importació JSON. Vull que, si l'arxiu és invàlid o té una estructura inesperada, mostre un error i conserve les dades actuals. No vull que un import fallit deixe l'app buida.

Prompt útil:

> Afig persistència amb localStorage. No afegis backend ni dependències. Inclou també exportació/importació JSON per no deixar les dades atrapades.

La simplicitat és bona quan també té una porta d'eixida.

Una ferramenta senzilla no és una ferramenta pobra si diu la veritat sobre els seus límits. La pobresa real és atrapar una persona dins d'una solució que semblava fàcil i no li deixa marxar.

`supabase start` — la fila que no vaig copiar

El dia que vaig decidir afegir base de dades, la meua aplicació deixà de ser una llibreta.

Fins eixe moment, `localStorage` m'havia servit. Però volia una cosa més incòmoda: obrir els recursos des de l'ordinador del centre, des del portàtil de casa i, potser, compartir una col·lecció amb altres docents.

Això ja no era "guardar en el navegador".

Era sincronitzar.

Juny acabava de començar, i el meu termini de setembre havia deixat de ser una abstracció: si la ferramenta havia de servir el primer dia de curs, esta era la peça que no podia ajornar.

La IA em va parlar de Supabase, Firebase i altres servicis. Tots prometien una entrada relativament amable al backend: base de dades, autenticació, APIs, panell web. També prometien una cosa que cal llegir amb atenció: facilitat.

La facilitat és meravellosa fins que et fa saltar preguntes.

Qui pot llegir les dades?

Qui pot escriure?

Què passa si publique la clau equivocada?

Què passa si el projecte creix?

Vaig demanar a la IA que no escriguera codi encara. Volia un esquema:

> Explica'm què necessite per passar de localStorage a Supabase. No escrigues codi. Vull entendre taules, permisos i autenticació.

La resposta va ordenar el problema. Necessitava una taula `resources`, camps per a títol, URL, matèria, nivell, etiquetes i data. Necessitava decidir si hi hauria usuaris. Necessitava polítiques de seguretat. Necessitava variables d'entorn.

No era només canviar on es guardava l'array.

Era canviar el contracte de l'aplicació.

Eixa frase em va frenar. Una base de dades no és un calaix més gran. És un espai compartit amb normes. Si no poses normes, algú altre les patirà.

Per a provar-ho de veritat, vaig obrir un full de càlcul que tenia per casa: la llista d'enguany, amb noms, nivells i alguna observació que m'havia apuntat al marge —"repassar lectura", "conflicte amb el company de taula"—, coses que un docent escriu sense pensar que algun dia podrien acabar en una taula amb clau pública. Vaig arribar a copiar la primera fila.

Em vaig aturar amb els dits damunt del teclat.

De qui eren estes dades? On viuria eixe servidor, en quin país, sota quina llei? Si algú em preguntara a una reunió de pares per què el nom del seu fill estava en un núvol que jo no controlava del tot, quina resposta donaria? "Perquè anava més ràpid" no és una resposta que puga dir en veu alta.

Vaig tancar el full de càlcul.

Vaig ajornar l'autenticació i vaig crear una prova controlada amb dades no sensibles. La IA va generar el client, les consultes bàsiques i la migració conceptual. Jo vaig revisar especialment les polítiques. No perquè fora expert, sinó perquè entenia prou per a saber que allí no es podia anar a cegues.

El primer recurs guardat en Supabase va aparéixer en el panell web. Vaig recarregar. Continuava. Vaig obrir en un altre navegador. També.

Eufòria, sí.

Però menys innocent que amb `localStorage`.

Ara les dades ja no vivien només en ma casa.

I quan les dades ixen de casa, apareixen les claus. El projecte acabava d'estrenar un arxiu nou, menut i amb nom de secret.

Apunt tècnic

Abans d'afegir una base de dades, respon:

- Quines dades guardaré?
- Són sensibles?
- Necessite usuaris?
- Qui pot llegir?
- Qui pot escriure?
- Com exportaré les dades si canvie de servici?
- Si treballe en un centre educatiu: un nom, un curs i una observació ja són dades sensibles, encara que semblen innòcues en un full de càlcul.

Taula mínima

> resources - id - title - url - subject - level - tags - created_at

No és només codi

Amb servicis com Supabase o Firebase has de pensar en:

- variables d'entorn;
- claus públiques i privades;
- regles o polítiques de seguretat;
- còpies/exportació;
- límits del pla gratuït.

Prompt útil:

> Vull passar de localStorage a una base de dades. Abans d'escriure codi, ajuda'm a decidir si realment cal. Compara localStorage, fitxer JSON exportable i Supabase per a este cas.

La millor base de dades és la que respon a una necessitat real, no a una aspiració de semblar més professional.

`.env` — una barra baixa a les 23:40

El fitxer `.env` té nom de cosa secreta i, per una vegada, el nom no exagera.

Va aparéixer quan vaig connectar la base de dades. La IA em va dir que guardara allí la URL del projecte i la clau pública.

```
VITE_SUPABASE_URL=...          VITE_SUPABASE_ANON_KEY=...
```

El nom `ANON_KEY` em va tranquil·litzar massa. "Anon" sonava a clau sense importància. Després vaig llegir millor: algunes claus poden ser públiques si les polítiques de la base de dades estan ben configurades; altres no haurien d'eixir mai del servidor. La frontera no és intuïtiva.

El perill no és només pujar una contrasenya. És pujar una clau que, combinada amb permisos mal posats, deixe la porta oberta.

Vaig revisar `.gitignore`.

.env .env.local

Després vaig mirar `git status` com qui comprova si ha apagat el gas.

El `.env` no apareixia. Bé.

Però hi havia una altra pregunta que jo encara no m'havia fet: si el `.env` no puja a Git, com arriba eixa configuració al lloc publicat?

En local, el fitxer era una nota privada dins del projecte. En producció, el hosting necessitava les mateixes variables, però no el mateix fitxer. Vaig descobrir que Netlify, Vercel o qualsevol servici semblant tenen el seu propi apartat de variables d'entorn. Allí poses la URL, les claus públiques necessàries i qualsevol configuració que el build haja de conéixer.

La IA m'ho va explicar amb una frase que em va costar pair:

> En una app Vite, les variables que comencen per VITE_ poden acabar incloses en el JavaScript que rep el navegador.

O siga: `VITE_` no vol dir "segur". Vol dir "disponible per al frontend".

Eixa distinció em va ordenar el cap. Hi ha variables locals que només necessita el meu ordinador. Hi ha variables de build que el navegador pot conéixer. I hi ha secrets que no haurien d'arribar mai al navegador, encara que estiguen dins d'un `.env` perfectament ignorat per Git.

Vaig obrir el panell del hosting i vaig copiar les variables una per una. Després vaig fer un deploy. Fallava.

No per seguretat. Per ortografia.

Havia escrit:

```
VITE_SUPABASE_ANONKEY
```

En el codi, en canvi, llegia:

```
import.meta.env.VITE_SUPABASE_ANON_KEY
```

Una barra baixa. Una sola. El tipus d'error que no mereix cap gran teoria i que, precisament per això, pot fer-te perdre una vesprada.

Des d'aquell dia vaig afegir al README una secció que abans em semblava excessiva:

```
## Variables d'entorn

Crea un fitxer .env.local amb:
```

```
VITE_SUPABASE_URL=...  VITE_SUPABASE_ANON_KEY=...
```

En producció, configura les mateixes variables en el hosting.

El README no és només per a altres persones. També és per a recordar-te quina paraula exacta vas triar quan encara tenies el cap fresc.

Però jo ja havia escrit una clau en un exemple dins del README. La vaig llevar. També vaig buscar:

```
rg "SUPABASE|API_KEY|SECRET|TOKEN"
```

Vaig trobar una clau de veritat, enganxada en un commit de fa tres setmanes, dins d'un exemple de codi que havia copiat directament del xat sense mirar-lo dues vegades. El repositori era públic.

Vaig mirar l'hora: les 23:40. Vaig entrar al panell de Supabase, vaig revocar la clau amb el cor una mica accelerat pel que no havia passat encara —no hi havia proves d'ús estrany, el projecte tenia quatre gats—, i en vaig generar una de nova. Vaig actualitzar `.env`, vaig comprovar que l'aplicació tornava

a funcionar i vaig fer un commit amb un missatge sobri, sense explicar per què.

No va passar res greu. Però la lliçó no depenia que passara: depenia de saber que podia haver passat, i que jo no me n'havia adonat fins que vaig buscar-ho expressament. Els secrets no només es filtren en `.env`. També es filtren en captures, logs, commits, exemples i presses.

La IA pot generar codi molt ràpid amb claus enganxades en el lloc equivocat. I si li pegues una clau en el prompt, pot quedar en la conversa. La comoditat té memòria.

Des d'eixe dia, quan una eina em demana una clau, pare. Abans de copiar-la, pregunte: esta clau és pública, privada, revocable, limitada? Què passa si algú la veu?

No sempre sé la resposta. Però ja sé que la pregunta és obligatòria.

Amb les claus sota control, vaig pensar que ja podia ensenyar l'aplicació a algú. Abans, però, calia que es poguera mirar en un mòbil sense passar vergonya.

Apunt tècnic

Un `.env` guarda variables d'entorn locals.

Exemple:

```
VITE_SUPABASE_URL=https://exemple.supabase.co
VITE_SUPABASE_ANON_KEY=...
```

En Vite, només les variables amb prefix `VITE_` són accessibles des del codi del navegador:

```
const url = import.meta.env.VITE_SUPABASE_URL;
```

Això és útil, però també és un advertiment: si el navegador pot llegir-ho, qualsevol usuari avançat també pot veure-ho.

Protegir-lo

En `.gitignore`:

```
.env .env.local
```

En producció, no puges el fitxer `.env`: configura les variables en el panell del hosting.

Classifica les variables

- **Configuració pública**: URL d'un projecte, clau anon pensada per a frontend, nom d'entorn.
- **Secret real**: tokens privats, service role keys, contrasenyes, claus d'administració.
- **Local només**: camins del teu ordinador, ports, proves personals.

Una regla pràctica: si et molestaria veure-ho en la consola del navegador d'una persona desconeguda, no ha d'estar en una variable `VITE_`.

Buscar secrets abans de commit

```
rg "API_KEY|SECRET|TOKEN|PASSWORD|SUPABASE|FIREBASE"
```

També pots mirar què està a punt d'entrar en el commit:

```
git diff --cached
git status
```

Si ja has pujat una clau

1. Revoca o rota la clau en el servici.
2. Elimina-la del codi.
3. Revisa l'historial si el repo és públic.

4. No assumisques que esborrar el fitxer ho arregla tot.

Checklist abans de desplegar

- Les variables existixen en local.
- Les mateixes variables existixen en el hosting.
- Els noms coincideixen exactament.
- No hi ha secrets reals amb prefix de frontend.
- El README explica quines variables cal crear sense revelar-ne el valor.

Prompt útil:

> Revisa si este canvi pot exposar secrets. Indica quines variables han d'anar en .env, quines poden ser públiques i quines no han d'arribar al navegador.

La confiança digital no es perd només amb grans catàstrofes. També es perd amb una clau copiada sense pensar, amb una variable mal posada, amb la idea perillosa que allò que és còmode és automàticament innocu.

`responsive` — deu segons al pati

La meua aplicació era preciosa fins que la vaig obrir en el mòbil.

En el portàtil, les targetes respiraven. Els filtres estaven alineats. El botó d'exportar tenia una dignitat discreta. En el mòbil, tot allò es convertia en una cua de supermercat: camps estrets, botons partits, textos que semblaven demanar perdó.

Vaig sentir una decepció injusta. La IA havia fet un disseny "responsive", o això deia. Però responsive no és una paraula màgica. És una promesa que cal provar.

Vaig obrir les eines de desenvolupador del navegador i activí la vista mòbil. L'aplicació va confessar immediatament. El formulari tenia tres columnes quan en necessitava una. Les targetes no tenien prou espai. Un botó amb text llarg desbordava.

Li vaig passar una captura a la IA i vaig escriure:

> No canvies l'estil general. Adapta esta pantalla a mòbil: - formulari en una columna; - botons que no desborden; - targetes llegibles; - mantindre contrast.

La captura va ajudar més que tres paràgrafs. Els bugs visuals no sempre apareixen en la consola. De vegades cal ensenyar-los.

Després del canvi, la pantalla no era més espectacular. Era més usable. I eixa diferència em va importar. Una aplicació no està acabada quan es veu bé en el teu monitor, amb la teua resolució i la teua paciència. Està més prop d'estar acabada quan una altra persona pot usar-la en condicions pitjors.

També vaig provar el contrast. Alguns grisos que en el portàtil pareixien elegants en el mòbil eren tímids fins a la desaparició. La IA tendeix a estimar els grisos subtils. Els ulls cansats, no tant.

Vaig pujar contrast, vaig ampliar espais, vaig fer que els botons ocuparen tota l'amplada en mòbil. Menys fi, més útil.

Dilluns, al pati, li vaig ensenyar el mòbil a Empar.

—Prova-ho amb el teu, no amb el meu —li vaig dir, i li vaig passar l'enllaç de la xarxa del centre, el que Vite deixa obert quan arrenques amb `--host`.

Ella el va obrir sense preguntar res, com qui obri qualsevol cosa que li passen. Va fer un parell de tocs, va arrufar el nas i em va tornar el mòbil.

—El botó d'afegir es queda amagat davall del teclat.

Jo no ho havia vist. En les eines de desenvolupador, el teclat no existix; és una simulació sense saliva ni presses. Al pati, amb el mòbil real, amb el teclat real ocupant mig de la pantalla i la campana a punt de sonar, el defecte era evident.

—I ara ho arregles? —em va preguntar, mig de conya.

Li vaig dir que sí, encara sense saber com. Ho vaig resoldre eixa vesprada amb un marge de desplaçament i una prova amb el teclat obert de veritat, no simulat. Però la lliçó important no era eixa: jo podia simular un mòbil dotze hores seguides sense trobar el que Empar va trobar en deu segons, perquè ella no estava provant l'aplicació. L'estava usant.

El disseny no és decorar.

És llevar excuses al mal ús.

Empar havia provat la versió del pati per la xarxa del centre, amb el meu portàtil fent de servidor. La pregunta següent era inevitable: i si li donava una adreça de veritat?

Apunt tècnic

Checklist responsive mínim:

- Provar amplades menudes.
- Formularis en una columna en mòbil.
- Botons amb altura suficient.
- Text sense desbordar.
- Contrast llegible.
- Espais tàctils còmodes.
- Provar en un dispositiu real, no només en eines de desenvolupador (per exemple exposant el servidor local amb `--host`).

CSS típic:

```
@media (max-width: 640px) {
  .formulari {
    grid-template-columns: 1fr;
  }
```

```
  button {
    width: 100%;
  }
}
```

Prompt útil:

> Adapta esta pantalla a mòbil sense canviar l'estil general. Prioritza llegibilitat, contrast i que cap text desborde. Et passe captura de la vista actual.

Si pots, dona captures. Per a la IA, una imatge d'un botó trencat pot valdre més que "es veu raro".

Dissenyar és acceptar que l'altre no usarà la teua ferramenta en les condicions ideals de la teua imaginació. Si una aplicació només funciona quan tot acompanya, encara no ha conegut el món.

`deploy` — una URL i el botó mig tallat

Publicar una aplicació és una manera molt ràpida de descobrir que encara no l'havies acabada.

En local, la meua biblioteca de recursos ja tenia una dignitat raonable. Guardava dades, exportava CSV, es veia decent en mòbil i passava el build. Era l'última setmana de curs, i jo volia que Empar se n'anara de vacances amb un enllaç en el mòbil: si al setembre havia de ser una ferramenta, este estiu havia de deixar de ser un secret. Vaig decidir pujar-la a un hosting estàtic.

La IA em va donar tres opcions: GitHub Pages, Netlify i Vercel. Vaig triar la que em semblà més directa per a aquella versió. No volia una arquitectura, volia una URL.

El primer deploy va ser emocionant. Una barra de progrés, uns logs, un missatge de "success" i, finalment, una adreça pública. La vaig obrir com qui obri una finestra després de pintar una habitació.

Funcionava.

Durant uns minuts.

Després vaig començar a provar amb ulls de persona desconfiada. Obrir en incògnit. Obrir en el mòbil. Enviar l'enllaç a un altre navegador. Recarregar. Tornar arrere. Afegir un recurs. Exportar. Mirar la consola.

Vaig enviar l'enllaç a Empar amb una frase prudent:

> No és definitiu, però pots provar si obri bé?

La prudència era nova. Abans hauria escrit "ja funciona". Ara sabia que "funciona" és una paraula que necessita cognoms: funciona en local, funciona en producció, funciona en mòbil, funciona sense les meues dades guardades, funciona quan una altra persona entra per primera vegada.

Als deu minuts, Empar em contestà:

> Obrir obri, però en el mòbil el botó d'afegir queda mig tallat.

El deploy havia fet una cosa meravellosa i cruel: havia tret el projecte del meu escenari perfecte. En

el meu portàtil, amb la finestra ampla i el zoom que jo use, tot semblava ordenat. En el seu mòbil, amb una barra del navegador més alta i el teclat apareixent quan tocava un camp, la pantalla contava una altra història.

Vaig obrir les eines de desenvolupador, vaig activar la vista mòbil i vaig veure-ho. El botó no estava "mig tallat" per cap maledicció del hosting. Jo havia fixat una altura massa optimista en un contenidor. En el món real, les pantalles no demanen permís per ser més menudes.

La IA em va proposar una solució immediata. Esta vegada vaig demanar-li primer una explicació:

> Explica per què este layout falla en mòbil. Després proposa el canvi CSS mínim. No canvies colors ni estructura visual.

El canvi va ser menut. El guany, gran: vaig començar a provar el deploy com si fora una altra aplicació, no com una continuació automàtica del meu entorn local.

Publicar convertix les suposicions en fets. Allò que "després ja miraré" es posa davant de tothom.

La URL pública és una alegria, però també és una responsabilitat menuda: si algú entra, allò parla per tu.

Vaig afegir una checklist al projecte:

```
# Abans de publicar

- npm run build - provar en mòbil - revisar consola - comprovar rutes - revisar variables d'entorn - provar en finestra privada
```

No era sofisticada. Era útil.

També vaig aprendre que deploy no és una cosa única. Hi ha primer deploy, deploy de correcció, deploy de prova, deploy que desfàs perquè has sigut massa optimista. Publicar forma part del cicle, no és una cerimònia final.

El botó “Deploy” no t’absol.

Només et dona públic.

Després del segon deploy vaig fer una cosa que m’hauria semblat exagerada en el primer projecte: vaig escriure una nota de versió.

```
## 0.2.1
```

> - Correcció de layout en mòbil. - Revisada importació/exportació JSON. - Afegida checklist abans de publicar.

No era per solemnitat. Era per memòria. Quan una aplicació està viva, els canvis comencen a confondre's. La IA pot reconstruir part de la història a partir del Git, però el sentit del canvi —per què ho vas tocar— convé escriure'l quan encara el recordes.

També vaig crear una regla personal: no fer deploy si no podia quedar-me deu minuts després mirant-lo. Publicar i tancar el portàtil és una temptació forta, sobretot quan el build ha passat i el hosting diu "success". Però molts errors apareixen en els primers minuts: una variable que falta, una ruta trencada, un CSS que en mòbil respira malament, una consola plena d'avisos que en local havies deixat passar.

Eixa nit, mirant la URL des del mòbil, vaig recordar la pregunta que m'havia quedat penjada des del principi: ara que existix, què penses fer amb ella? Ja no era retòrica. Tenia una adreça, i qualsevol podia obrir-la.

Qualsevol que no fora jo, per exemple. I ahí començava una classe de problemes que el meu ordinador no em podia ensenyar.

Apunt tècnic

Abans de desplegar:

```
npm run build
```

Comprova:

- que no hi ha errors de build;
- que les rutes funcionen;
- que les variables d'entorn estan configurades en el hosting;
- que no publiques secrets;
- que el mòbil es veu bé;
- que la consola no mostra errors greus;
- que una persona sense dades prèvies pot usar-la;
- que pots desfer o corregir ràpid si alguna cosa falla.

Després de desplegar:

- obri la URL pública, no només el preview local;
- prova en finestra privada;

- prova una pantalla mòbil;
- mira la consola i la pestanya Network;
- fes una acció real de principi a fi;
- escriu una nota curta del canvi si altres persones l'estan usant.

Prompt útil:

> Vull desplegar esta app. Ajuda'm a preparar una checklist per al meu stack actual. No em dones només passos del hosting: inclou verificacions abans i després.

Desplegar és passar de "en el meu ordinador funciona" a "ara ho pot comprovar algú més".

Publicar no és el final de la intimitat amb el projecte. És el moment en què la intimitat es convertix en responsabilitat pública: allò que abans només et fallava a tu ara pot interrompre el dia d'una altra persona.

`It works on my machine` — el meu ordinador era còmplice

La frase “en el meu ordinador funciona” hauria d’aparéixer en roig en tots els teclats.

La vaig dir, naturalment.

Empar va obrir la meua aplicació publicada i em va escriure:

> No em carrega els recursos d'exemple.

Jo la vaig obrir en el meu portàtil. Funcionava. Vaig recarregar. Funcionava. Vaig fer el gest mental de declarar el problema resolt per votació individual.

Per sort, ja havia aprés a desconfiar de mi.

En finestra privada, fallava. En el mòbil, fallava. En producció, la ruta del fitxer JSON d’exemple estava mal construïda. En local, Vite l’havia resolta amb una amabilitat que el hosting no compartia.

El bug no era complicat. El diagnòstic sí, perquè el meu entorn estava ple de memòria: caché, dades en `localStorage`, servidor local, sessions obertes. El meu ordinador no era un jutge neutral. Era còmplice.

Vaig escriure a la IA:

> En local funciona, en producció no. No proposes solució encara. Fes-me una llista de diferències possibles entre els dos entorns.

La llista va ser més útil que qualsevol pegat: rutes, variables, caché, majúscules, build, permisos, dades locals, configuració del hosting.

Vaig netejar `localStorage`, vaig obrir incògnit, vaig revisar la pestanya Network i vaig trobar el 404 del JSON.

Un `404` no sempre significa que falte una pàgina. A vegades significa que tu i el servidor no enteneu igual la paraula "ací": en local, moltes eines resolen rutes i majúscules amb una amabilitat que el hosting no comparteix, i una aplicació d'una sola

pàgina pot simular una navegació interna que el servidor no sap reproduir si hi entres directament.

Corregir-ho va costar cinc minuts.

Arribar a mirar el lloc correcte, una hora.

La ruta era així:

```
fetch("/data/recursos.json")
```

En local, perfecta. En producció, l'aplicació vivia dins d'una subcarpeta del hosting. La barra inicial deia "busca des de l'arrel del domini", però el fitxer no estava allí. Estava davall del camí del projecte. Una barra. Una sola. Igual que la variable d'entorn mal escrita, el bug era minúscul i la pèrdua de temps, perfectament adulta.

La IA em proposà usar una ruta relativa:

```
fetch("./data/recursos.json")
```

Abans d'acceptar-ho li vaig demanar que m'explicara la diferència entre `/data` i `./data`. No per convertir-me en especialista en rutes, sinó per no quedar-me amb un conjur. Si el mateix problema tornava amb imatges, fonts o arxius d'ajuda, volia reconéixer-lo.

Després vaig provar tres URLs:

https://.../app/
https://.../app/data/recursos.json
https://.../app/una-ruta-interna

La segona era la clau. Si l'arxiu JSON no obria directament, l'aplicació tampoc tenia cap motiu per trobar-lo. A vegades diagnosticar és llevar capes fins que només queda una petició HTTP que diu sí o no.

Vaig afegir al README una secció menuda:

Com provar producció

1. Obrir la URL pública en finestra privada. 2. Obrir directament /data/recursos.json. 3. Provar una ruta interna recarregant la pàgina. 4. Revisar consola i Network.

Era una nota per a mi, però també una confessió: el meu ordinador ja no bastava com a prova.

Eixe dia vaig decidir que "funciona en el meu ordinador" no seria mai més una conclusió. Com a molt, seria el primer indici.

Mentre apuntava tot allò per a no oblidar-ho, vaig descobrir que el projecte tenia un altre arxiu

abandonat que també parlava de mi. I no deia res bo.

Apunt tècnic

Quan alguna cosa funciona en local i falla en producció, compara:

- rutes relatives;
- variables d'entorn;
- majúscules/minúscules en noms d'arxiu;
- caché del navegador;
- dades en `localStorage`;
- logs del hosting;
- pestanya Network del navegador;
- diferències entre `npm run dev` i `npm run build`;
- en aplicacions SPA, si falta configuració de fallback per a rutes internes;
- base path incorrecte en subcarpetes o GitHub Pages.

Prova:

- finestra privada;
- altre navegador;

- mòbil;
- netejar `localStorage`;
- URL directa de l'arxiu que falla;
- URL pública copiada tal qual, no la del servidor local;
- recarregar una ruta interna si uses una SPA.

Prompt útil:

> Funciona en local però falla en producció. Fes una taula amb causes possibles, com verificar cada una i quin canvi mínim provaries.

Prompt per a rutes:

> Explica la diferència entre estes rutes en una app desplegada en una subcarpeta: /data/recursos.json ./data/recursos.json data/recursos.json
>
> Indica quina convé usar i com provar-ho en producció.

"En el meu ordinador funciona" és només una forma educada de dir "encara no ho he exposat prou a la realitat". Aprendre amb IA també és aprendre a no confondre el teu entorn amb el món.

`README.md` — l'aparador amb la porta tancada

El README no és per als altres.

O no només.

El README és per a tu, d'ací tres setmanes, quan tornes al projecte amb menys context, menys entusiasme i una vaga sensació que "açò arrancava d'alguna manera".

Em va passar amb la biblioteca de recursos. Havia deixat el projecte quiet uns dies. En tornar, no recordava si calia crear `.env`, si les dades d'exemple venien d'un JSON o de Supabase, ni quin comando usava per previsualitzar el build.

Vaig mirar el README.

Feia vergonya.

Tenia un títol i una frase heroica:

```
# Recursos Aula

Aplicació per gestionar recursos educatius.
```

Molt bé. I ara què?

La primera temptació va ser la de sempre: delegar-ho sencer.

> Genera un README complet i professional per a este projecte.

El resultat feia goig. Tenia insígnies de colors a dalt de tot, un índex amb enllaços interns, una secció de "Características" amb emojis, instruccions de desplegament amb Docker, una guia de contribució amb normes per a pull requests, una taula de compatibilitat de navegadors, un apartat de "Roadmap" amb funcions que jo no havia planificat mai —"soporte multiusuario", "aplicación móvil nativa"— i, al final, una línia que deia:

> ## Licencia
>
> Este proyecto está bajo licencia MIT.

Cent quaranta línies. Semblava el README d'un projecte amb quinze desenvolupadors i una comunitat darrere. El meu tenia un desenvolupador que a estones tampoc hi era.

I quasi tot era mentira. No hi havia Docker. No hi havia guia de contribució perquè no hi havia contribuents. La taula de navegadors era decorativa: jo només havia provat en dos. El roadmap prometia coses que jo no pensava fer. I la llicència MIT, directament, no existia: cap arxiu LICENSE en tot el repositori. La IA no havia documentat el meu projecte; havia escrit el README estadísticament més probable.

Un README que promet més del que hi ha no és documentació. És màrqueting accidental. I es paga: la primera persona que seguira les instruccions de Docker es trobaria un mur, i jo quedaria com el venedor d'un pis ensenyant fotos d'un altre pis.

Vaig esborrar-ne tres quartes parts i vaig reescriure el que quedava com si li parlara a algú que volia usar el projecte sense llegir-me la ment.

Què és.

Com s'instal·la.

Com s'arranca.

Quines variables necessita.

Què funciona.

Què falta.

Com es desplega.

La documentació no va fer el projecte més avançat. El va fer més habitable. I, sobretot, va millorar les converses amb la IA. Quan l'agent podia llegir un README clar, deixava d'inventar tant.

Un README és una forma de memòria compartida entre tu, el teu jo futur i les eines que t'ajuden.

El README ordenava la memòria del projecte. Em faltava ordenar la memòria de les meues pròpies idees.

La meua llista de millores havia començat en un paper.

Després passà a un missatge guardat.

Després a tres notes soltes.

Després al caos.

Volia afegir etiquetes de colors, importar JSON, filtrar per nivell, millorar el mòbil, revisar accessibilitat, afegir una pàgina d'ajuda i canviar el nom de l'app. Tot alhora. La IA, si li dones una llista així, pot intentar ajudar-te amb tot alhora. I eixa és una manera elegant de multiplicar problemes.

Vaig obrir issues.

Una per idea.

No per fer-me professional, sinó per reduir soroll. Cada issue tenia tres parts:

```
## Objectiu

## Criteri d'acceptació

## Notes
```

La màgia no estava en GitHub ni en l'etiqueta "issue". Estava en obligar-me a escriure què voldria considerar acabat.

"Millorar filtres" no és una tasca.

"Afegir filtre per nivell i comprovar que es combina amb cerca de text" sí.

Les issues també milloraren els prompts. En lloc de demanar a la IA que "millorara la app", li passava una issue concreta. L'agent ja no havia d'endevinar prioritats. Havia de resoldre un encàrrec.

El projecte deixà de ser una massa de possibilitats i es convertí en una cua de treball.

Molt menys romàntic.

Molt més útil.

La primera issue que no vaig escriure jo va arribar al cap d'uns dies.

Un docent de no sé on —el perfil tenia el logotip d'un institut i tres repositoris— preguntava si podia adaptar l'aplicació per al seu centre. I tancava amb una frase que em va semblar de tràmit:

"Quina llicència té el projecte?"

Cap, vaig pensar. És públic. Agafa'l.

Per si de cas, abans de contestar li ho vaig preguntar a la IA.

Sort.

Resulta que "sense llicència" no vol dir "per a tothom". Vol dir el contrari. Si un repositori no té un arxiu LICENSE, per defecte tots els drets són de l'autor. La gent pot mirar el codi, perquè és visible, però legalment no pot copiar-lo, modificar-lo ni reutilitzar-lo.

Jo pensava que "públic" volia dir "lliure".

Volia dir "visible". Com un aparador amb la porta tancada.

Tenia un repositori obert a tot el món que ningú podia usar. La generositat d'ensenyar-ho tot i la lletra menuda de no deixar tocar res, en el mateix gest.

Li vaig preguntar a la IA quina llicència posar. Em va fer tres preguntes: si volia que altres pogueren usar el codi comercialment, si volia obligar que les modificacions també foren obertes, i si m'importava més compartir o controlar. Vaig triar MIT: fes el que vulgues, no em faces responsable.

La mateixa que aquell README inflat m'havia atribuït per error setmanes abans. La IA havia documentat un projecte que encara no existia; ara, almenys, una de les seues mentires s'havia fet veritat.

Un arxiu. Vint línies que tampoc vaig escriure jo.

Vaig contestar la issue: "Ja té llicència. Tot teu."

README i LICENSE viuen junts a l'arrel del repositori per alguna raó. Un explica com s'usa el projecte. L'altre diu si tens permís per usar-lo. Havia passat setmanes polint el primer sense saber que el segon existia.

El docent va clonar el repositori aquell mateix dia. Jo encara no sabia que aquell clon tornaria a aparéixer en esta història, per una raó molt pitjor que una llicència.

Apunt tècnic

Estructura mínima d'un README

Nom del projecte

Descripció curta.

Estat

Què funciona i què està pendent.

Instal·lació

“‘bash npm install “‘

Desenvolupament

“‘bash npm run dev “‘

Build

“‘bash npm run build “‘

Variables d'entorn

Explicar quines calen, sense posar secrets.

Desplegament

Passos o enllaç al hosting.

Prompt útil:

Revisa este README pensant en algú que obri el projecte d'ací un mes. Què falta per poder instal·lar, arrancar i verificar l'app?

Una bona issue

Una bona issue inclou:

- títol concret;
- problema o objectiu;
- criteri d'acceptació;
- límits;
- passos de verificació.

Exemple:

Afegir filtre per nivell

Objectiu

Poder filtrar recursos per nivell educatiu.

Criteri d'acceptació

- selector de nivell visible; - es combina amb la cerca de text; - opció "Tots"; - npm run build passa.

Prompt útil:

> Converteix esta idea en una issue accionable amb criteris d'acceptació. Després proposa un pla de canvi mínim.

Llicències habituals

- **MIT**: qualsevol pot usar, copiar i modificar el codi, també comercialment, mantenint l'avís de copyright. La més curta i la més comuna.
- **Apache 2.0**: pareguda a MIT, amb protecció explícita de patents. Habitual en projectes grans.
- **GPL**: qui modifique i redistribuïsca el codi ha de mantindre'l obert amb la mateixa llicència.
- **Sense arxiu LICENSE**: tots els drets reservats, encara que el repositori siga públic. El codi es pot mirar, però ningú pot reutilitzar-lo legalment.

Prompt útil:

> Vull publicar este projecte en un repositori públic. Vull que [qualsevol el puga usar / les modificacions continuen sent obertes / no s'use comercialment]. Qui-

na llicència em convé i per què? Genera l'arxiu LICENSE.

Documentar és una manera de cuidar l'absència. Escrius perquè demà no estaràs tan fresc, perquè algú arribarà sense la teua memòria, perquè una IA llegirà el projecte sense haver viscut el camí que t'ha portat fins ací.

`git push` — nou hores amb la porta oberta

Jo pensava que esta lliçó ja l'havia pagada.

Quan vaig muntar el `.env`, havia trobat una clau antiga en un commit, l'havia revocada i m'havia promés vigilar. Des d'aleshores mirava `git status` com qui comprova el gas. Em creia una persona escarmentada.

Però els escarmentats tenim un punt cec: només vigilem la porta per on ja ens han robat.

Va passar amb un script. Volia omplir la base de dades amb cinquanta recursos d'exemple per a provar els filtres amb volum de veritat, i les polítiques de seguretat —les mateixes que jo havia revisat amb tanta cerimònia— no em deixaven inserir-los en bloc. La IA em va donar la solució amb tota naturalitat:

> Per al seed pots usar la service_role key. Es salta les polítiques. Usa-la només en local, mai en el navegador.

Només en local. Vaig apegar la clau directament dins de `scripts/seed.js`, perquè el `.env` em semblava una cosa de l'aplicació i allò era un script d'anar per casa. Va funcionar a la primera. Cinquanta recursos. Content amb els meus filtres, a les dotze de la nit vaig fer un commit d'eixos que ho arrepleguen tot:

```
git add .
git commit -m "Neteja i scripts"
git push
```

L'endemà de matí tenia un correu de GitHub. Assumpte: possible secret detectat al teu repositori.

El cos del correu era educat i concret. Nom de l'arxiu, número de línia, tipus de clau. Com una multa amb fotografia.

El primer que vaig fer va ser el més humà i el més inútil: esborrar.

```
git rm scripts/seed.js
git commit -m "Esborra script temporal"
git push
```

Vaig recarregar la pàgina del repositori. L'arxiu ja no estava. Vaig respirar.

La calma em va durar el que vaig tardar a preguntar-li a la IA si ja estava resolt.

No ho estava. Git no oblida: l'arxiu havia desaparegut de l'última foto, però totes les fotos anteriors continuaven allí, públiques, amb la clau dins. Qualsevol podia obrir l'historial i llegir-la. I encara pitjor: el `push` de la nit anterior ja havia repartit còpies. Vaig pensar en el docent de l'institut, el de la llicència, que tenia un clon sencer del meu repositori en el seu ordinador. Esborrar la pàgina del diari no servix de res quan el diari ja s'ha fotocopiat.

Vaig mirar les hores. Entre el push nocturn i el correu: nou hores. Nou hores amb la clau mestra de la meua base de dades a la vista de qualsevol. I era la mestra de veritat: l'altra vegada s'havia filtrat una clau pública i vella; esta se saltava totes les polítiques que jo havia configurat. La porta blindada, amb la clau posada per fora.

Vaig notar la temptació de fer moltes coses alhora i totes de pressa. En lloc d'això, vaig escriure:

> S'ha filtrat la service_role key en un repositori públic. Dona'm un pla per orde

de prioritat. No vull fer res de pressa que empitjore les coses.

El pla va arribar sense dramatisme, i l'orde era la part important.

Primer, rotar la clau. Abans de netejar res, abans d'investigar res. Mentre la clau vella fora vàlida, tot lo altre era decorar. Vaig entrar al panell, la vaig regenerar, i la que estava escampada per l'historial es va convertir en paper mullat. Tres minuts.

Segon, comprovar que l'aplicació seguia funcionant amb la nova. Funcionava.

Tercer, mirar si algú havia usat la clau durant aquelles nou hores. Els registres no mostraven res estrany. El projecte continuava tenint quatre gats, i per una vegada això era una bona notícia.

Quart, decidir què fer amb l'historial. Es podia reescriure amb ferramentes fetes per a això, avisant abans a qui tinguera clons. Però una clau revocada en l'historial ja no és una porta: és l'anècdota d'una porta. Vaig deixar la neteja apuntada com a issue i vaig dedicar l'energia al que sí que importava: que no tornara a passar.

Cinqué, els hàbits. El script de seed va renàixer llegint la clau d'una variable d'entorn, el `.gitignore` va guanyar una línia per als meus experiments, i jo vaig guanyar una norma nova: els scripts d'anar per casa són codi, i el codi acaba en Git encara que jures que no.

L'ansietat d'aquell matí no me la va llevar cap comando. Me la va llevar entendre la diferència entre amagar i revocar. Amagar és esperar que ningú mire. Revocar és que ja no importe qui mira.

L'agost avançava, el curs s'acostava, i el projecte havia sobreviscut al seu pitjor dia. Començava a semblar que allò mereixia un número de versió.

Apunt tècnic

Senyals que un secret s'ha filtrat

- Correu de GitHub o d'un servici de detecció (secret scanning).
- Una clau apareix buscant en el codi o en l'historial:

```
rg "API_KEY|SECRET|TOKEN|service_role"
git log -p -S "eyJ" -- .
```

- Activitat estranya en el panell del servici (peticions que tu no has fet).

Passos, en este orde

1. **Rota la clau primer.** Sempre. Una clau revocada deixa de ser un problema encara que estiga escrita en mil llocs.
2. Actualitza `.env` (i el hosting) amb la clau nova i comprova que l'aplicació funciona.
3. Revisa els registres del servici per si la clau s'ha usat mentre estava exposada.
4. **No confies en `git rm`.** Esborrar l'arxiu no l'esborra de l'historial, i el push ja ha repartit còpies i clons.
5. Si cal netejar l'historial (repositori amb més gent, requisits legals), usa `git filter-repo` o BFG Repo-Cleaner, i avisa abans a qui tinga clons: reescriure historial els trenca el repositori.
6. Revisa `.gitignore` i el costum que ha causat la filtració: scripts temporals, exemples del xat, captures.

La regla de les dues claus

- La clau **pública** (anon) pot viure en el navegador si les polítiques estan ben posades.
- La clau **de servici** (service_role) se salta les polítiques: no ha d'eixir mai del teu entorn local o del servidor, i mai d'un `.env` ignorat per Git.

Prompt útil:

> S'ha filtrat esta clau en un repositori públic: [tipus de clau, servici]. Dona'm un pla de contenció per orde de prioritat: què rotar primer, què comprovar després i què puc deixar per a més tard. Indica també què NO cal fer.

Amagar un secret és esperar que ningú mire. Revocar-lo és que ja no importe.

`v1.0.0` — dir que no la setmana abans del curs

La versió 1.0.0 no va ser la versió perfecta.

Va ser la primera que em va fer menys vergonya compartir. I tenia data de caducitat inversa: era l'última setmana d'agost, i la promesa que m'havia fet al principi —que Empar poguera obrir açò el primer dia de curs— ja no admetia pròrrogues.

La biblioteca de recursos ja tenia el que jo havia promés al principi: afegir recursos, etiquetar-los, buscar, filtrar, guardar, exportar, importar i obrir-la en el mòbil sense que els botons fugiren de la pantalla. Després de les proves amb Supabase i de l'ensurt de les claus, havia pres una decisió poc lluïda però honesta: la primera versió pública seria local i exportable. La sincronització quedava fora fins que jo poguera defensar-la bé.

No tenia login. No tenia col·laboració en temps real. No tenia analítica, ni panell d'administració, ni

cap d'eixes coses que la IA suggerix amb entusiasme quan detecta que encara tens vesprada lliure.

Tenia una funció clara.

Vaig crear una issue final:

> # Preparar v1.0.0
>
> - revisar README; - npm run build; - provar en mòbil; - netejar logs; - exportar dades de prova; - crear tag.

Després vaig afegir una línia que no semblava tècnica:

> - escriure limitacions conegudes.

Em va costar més que el build.

Una limitació escrita és una renúncia pública. Vol dir admetre que la ferramenta no fa algunes coses i que no les fa per decisió, no per oblit. Vaig obrir el README i vaig escriure:

> ## Limitacions
>
> - Les dades es guarden en este navegador. - No hi ha sincronització entre dispositius. - No s'han d'introduir dades per-

sonals d'alumnat. - La importació JSON substituïx la biblioteca actual.

Llegit així, el projecte pareixia menys impressionant. També pareixia més honrat.

La IA em va suggerir una frase més amable: "Pròximament sincronització entre dispositius". La vaig llevar. No sabia si seria pròximament. No volia convertir una possibilitat en promesa. Un README pot mentir amb futur simple.

La IA m'ajudà a revisar textos, detectar un `console.log` oblidat i simplificar una part del codi.

També em va proposar dues millores que no li havia demanat.

La primera: un sistema de comptes d'usuari amb inici de sessió, "per si en el futur vols que cada docent tinga la seua col·lecció privada". Sonava raonable, i durant uns segons em vaig imaginar explicant-li a algú "sí, té login", com si això certificara alguna cosa.

La segona: un panell d'estadístiques amb gràfiques dels recursos més visitats, "per a saber què funciona millor". També quedava bé sobre el paper.

Cap de les dues resolia un problema que tinguera de veritat. Ningú m'havia demanat comptes separats. Ningú necessitava gràfiques d'un projecte amb quatre usuàries. Eren millores pensades per a una aplicació més gran que la que jo tenia, no per a la que jo havia promés.

Vaig escriure:

> No. Cap de les dues toca l'objectiu de la v1.0.0. Ix de l'abast.

Res es va esfondrar. La IA no es va ofendre —no s'ofén mai—, i el projecte va continuar sent el que havia de ser: una ferramenta xicoteta que fa bé una cosa. Eixa va ser una satisfacció nova: dir que no a la IA sense sentir que perdia una oportunitat.

Vaig executar:

```
npm run build
git status
git add .
git commit -m "Prepara versio 1.0.0"
git tag v1.0.0
```

El tag era simbòlic, però els símbols importen. Deia: fins ací, açò és una versió.

Després vaig fer una cosa més: vaig descarregar l'export JSON de prova i el vaig guardar fora del projecte. Si la v1.0.0 era una versió, necessitava una manera de comprovar que la següent no trencava les dades d'esta. No era una bateria de tests sofisticada. Era un arxiu menut amb tres recursos, un sense matèria i un amb cometes en el títol. El meu petit fòssil.

recursos-test-v1.json

Cada vegada que tocara importació, exportació o estructura de dades, podria intentar importar aquell arxiu. Si fallava, no era només un bug: era una ruptura amb algú que ja havia confiat en la versió anterior.

No "la definitiva".

Una versió.

La diferència em va llevar pes. El programari no necessita nàixer complet. Necessita nàixer amb una forma honesta i una manera de continuar.

Quan vaig enviar l'enllaç a Empar i a un altre company —un dijous de finals d'agost, amb el curs a

tocar—, no vaig escriure “mira quina app he fet amb IA”. Vaig escriure:

“He fet una primera versió d’una ferramenta per guardar recursos. Proveu si vos servix i digueu-me què falla.”

Era menys espectacular.

Era molt més real.

Havia arribat al setembre amb una ferramenta, no amb una promesa. La resposta d’Empar, però, no va arribar aquell dia. Va arribar quan el curs ja havia arrancat, i no era un elogi.

Apunt tècnic

Checklist de primera versió:

- objectiu clar;
- README actualitzat;
- limitacions conegudes escrites;
- sense logs temporals;
- build correcte;
- prova en mòbil;
- prova en finestra privada;
- dades exportables;

- arxiu de prova exportat;
- commit final;
- tag opcional.

Comandos:

```
npm run build
git status
git add .
git commit -m "Prepara versio 1.0.0"
git tag v1.0.0
```

Guardar una exportació de prova:

> recursos-test-v1.json

No cal pujar dades reals. Al contrari: usa dades fictícies que representen casos útils.

Prompt útil:

> Ajuda'm a preparar una checklist de v1.0.0. No proposes funcions noves llevat que siguen imprescindibles. Prioritza estabilitat, documentació i verificació.

Prompt per a limitacions:

> Ajuda'm a escriure una secció de limitacions conegudes per al README. Ha de ser

clara i honesta, sense prometre futures funcions que encara no estan planificades.

Una primera versió no és una promesa de perfecció. És una promesa més difícil: açò fa el que diu, no amaga el que no fa i pot continuar sense trair la confiança de qui l'ha començada a usar.

`maintenance mode` — el diumenge d'Empar

El primer missatge d'una usuària real va arribar la segona setmana de curs. Era una frase curta, d'Empar:

> Estaria bé poder duplicar un recurs.

Vaig somriure amb una vanitat molt humana. Algú havia usat prou l'aplicació per a voler una millora. Després vaig sentir el pes contrari: ara la ferramenta ja no era només una prova meua. Havia entrat en la rutina d'algú.

Vaig pensar en Empar preparant la setmana un diumenge a la vesprada, cercant un recurs de plàstica per a repetir amb un altre grup, confiant que el botó de duplicar estaria allí dilluns. Si jo trencava alguna cosa sense adonar-me'n —un desplegament ràpid, una migració mal provada—, no seria jo qui ho patiria a mitjanit. Seria ella, amb la classe l'endemà i la ferramenta penjada.

Eixe pensament em va canviar la manera de publicar canvis. Ja no bastava que a mi em funcionara. Havia d'imaginar-me-la usant-ho sense mi al costat.

La IA hauria pogut afegir el botó en minuts. Però mantindre una aplicació no és respondre sí a cada petició. Vaig obrir una issue, vaig descriure el cas i vaig preguntar:

Duplicar recurs és realment necessari?

On apareix el botó?

Què passa amb les etiquetes?

El duplicat ha de dir "còpia"?

També vaig escriure el que no faria:

> Fora d'abast: - plantilles de recursos; - duplicació múltiple; - historial de versions; - sincronització entre usuaris.

Això em va salvar. La IA, si li deixes espai, pot convertir "duplicar un recurs" en un sistema complet de clonació, plantilles i confirmacions elegants. Tot podria ser útil algun dia. Eixe "algun dia" és una trampa molt cara.

Vaig fer una mini conversa amb Empar:

> Quan dius duplicar, vols copiar tot i després editar només el curs?
>
> Sí. Sobretot per no escriure de nou títol, enllaç i etiquetes.

La resposta va simplificar el problema. No volia una funció abstracta. Volia estalviar-se una repetició concreta.

Vaig fer el canvi menut, el vaig provar i el vaig publicar. Després vaig actualitzar el README.

Mantindre és això: canvis menuts, criteri, versions, escoltar sense obeir-ho tot.

La setmana següent arribà una altra petició:

> Podria tindre colors per matèria?

Esta era més seductora. Jo també volia colors. La IA podia fer-ho en cinc minuts. Però el projecte ja tenia filtres, etiquetes, importació, exportació i un CSS que començava a tindre zones delicades. Afegir colors no era només pintar: era decidir paleta, accessibilitat, contrast, què passava amb matèries noves, com s'exportava eixa informació i si el JSON antic continuava funcionant.

Vaig convertir la petició en una issue, però no la vaig fer.

No per menyspreu. Per salut.

Vaig contestar:

> Ho apunte. Abans vull estabilitzar importació/exportació i duplicar. Si ho fem, ha de mantindre bon contrast en mòbil.

Mantindre també és cuidar la confiança. Si cada petició es convertix en un canvi immediat, l'aplicació creix més ràpid que la teua capacitat d'entendre-la. I quan això passa, la persona que depén de la ferramenta no veu la teua bona voluntat; veu que un dilluns una cosa que funcionava ha deixat de funcionar.

També arribaren avisos de dependències. Paquets que volien actualitzar-se. Auditories. Versions noves. La IA podia aplicar-ho tot, però jo vaig aprendre a preguntar:

Què guanye?

Què arrisque?

És una actualització de seguretat o només novetat?

El manteniment té menys èpica que crear. També té més veritat. Una aplicació no demostra el seu valor el dia que naix, sinó les setmanes següents, quan continua sent comprensible.

Vaig obrir un arxiu molt simple:

```
# CHANGELOG

## 1.0.1

- Afegit botó per duplicar recursos. - Corregit missatge d'importació JSON invàlida.

## 1.0.0

- Primera versió pública.
```

No seguia cap estàndard perfecte. Servia. I servir és una virtut infraestimada en el programari personal.

El més difícil del manteniment no va ser aprendre comandos. Va ser acceptar que una ferramenta viva necessita ritme. Ni abandonar-la després de publicar-la ni convertir cada vesprada en una reforma integral. Ritme: escoltar, apuntar, agrupar, provar, publicar, mirar.

I, per damunt de tot, una disciplina que ja m'havia salvat més d'una vegada i que encara no he contat del tot: llegir abans de firmar.

Apunt tècnic

Hàbits de manteniment:

- issues per a peticions;
- canvis menuts;
- changelog simple;
- actualitzar README;
- revisar dependències amb calma;
- fer còpies/exportacions;
- no acceptar totes les idees;
- diferenciar errors, millores i capricis;
- provar amb dades reals però no sensibles;
- conservar compatibilitat amb dades exportades antigues quan siga possible.

Triar què fer

Per a cada petició:

- quin problema resol?;
- qui el patix?;

- hi ha una solució més simple?;
- què pot trencar?;
- com ho provaré?;
- què queda fora d'abast?

Dependències

Abans d'actualitzar:

```
npm outdated
npm audit
```

I pregunta què implica el canvi. Una actualització de seguretat urgent no és el mateix que una versió major que canvia APIs.

Prompt útil:

> Avalua esta petició d'usuari. Digues si encaixa amb l'objectiu de l'app, quin risc té i quina seria la implementació mínima.

Prompt per a dependències:

> Revisa estes actualitzacions de dependències. Separa seguretat, compatibilitat i novetats. Indica quines aplicaria ara i quines deixaria per a més avant.

Mantindre és protegir la forma del projecte mentre el deixes créixer.

Revisar abans de firmar — el canvi que quasi accepte a cegues

El primer canvi gran que em va fer la IA i que jo hauria aprovat sense llegir va ser una millora aparentment innocent, nascuda d'una petició del nou curs: ordenar els recursos per data i per matèria.

La petició era clara:

> Afig ordenació dels recursos. Vull poder ordenar per data de creació i per matèria. Mantín el disseny actual.

La IA va treballar amb una eficàcia que, vista des de fora, semblava professional. Afegí un selector, una funció d'ordenació i uns quants estils. El build passava. La pantalla es veia bé. En provar-ho ràpidament, els recursos canviaven d'ordre.

Tot convidava a fer commit.

Però ja havia arribat massa lluny en este llibre per confondre "sembla bé" amb "està bé".

Vaig obrir el diff.

```
git diff
```

I em vaig obligar a llegir-lo com si no l'haguera escrit una màquina brillant, sinó una persona cansada un divendres a última hora. La diferència és important. Quan lliges codi de la IA amb admiració, busques confirmació. Quan el lliges com una revisió, busques responsabilitat.

El primer detall era menut. La funció d'ordenació modificava l'array original:

```
recursos.sort((a, b) =>
  a.subject.localeCompare(b.subject));
```

Jo havia aprés prou React per sospitar d'això. Si `recursos` era estat, modificar-lo directament podia provocar comportaments estranys. No sempre esclataria. Pitjor: podia funcionar quasi sempre i fallar quan menys ganes tens de depurar.

Li vaig preguntar:

> Revisa este diff com si fores una revisora estricta. Busca mutacions d'estat, regressions possibles i casos límit. No proposes refactorització gran si no cal.

La IA, davant del seu propi codi, va ser més crítica que en la primera resposta. Va assenyalar la mutació i proposà copiar abans d'ordenar:

```
const recursosOrdenats = [...recursos].sort((a, b) =>
  a.subject.localeCompare(b.subject)
);
```

El segon detall era més humà. L'ordenació per matèria fallava quan un recurs no tenia matèria. No perquè l'aplicació es trencara sempre, sinó perquè alguns recursos antics, importats des d'un JSON anterior, no tenien eixe camp. Jo ho sabia. La IA no. O millor dit: la IA podia inferir-ho si jo li mostrava dades antigues, però no podia endevinar la història del projecte.

Vaig crear tres recursos de prova:

1. Amb matèria: Matemàtiques 2. Amb matèria: Llengua 3. Sense matèria

El tercer es comportava com un convidat sense cadira. A vegades quedava al principi, a vegades al final, segons com havia quedat la funció.

No era un gran drama. Era just el tipus de detall que convertix una millora en una futura queixa.

Vaig demanar criteri:

> Quin comportament seria més previsible per als recursos sense matèria?

La resposta em va semblar raonable: enviar-los al final i mostrar “Sense matèria” si calia. Vaig acceptar el canvi menut.

El tercer detall no estava en el codi. Estava en la interfície. El selector de l’ordre no explicava quin ordre estava actiu. Deia només:

> Ordenar

Per a mi, que acabava de crear-lo, era suficient. Per a una altra persona, no. Vam canviar-lo per opcions explícites:

> Més recents primer Matèria A-Z Matèria Z-A

Llavors sí. El canvi ja no sols funcionava en el meu cap.

Aquella vesprada vaig entendre una forma nova d’autoria. Jo no havia escrit la major part del codi. Però havia fet les preguntes que el convertien en

codi del meu projecte: què muta?, què passa amb dades antigues?, què veu l'usuari?, com ho prove?, com ho desfaig?

Firmar un canvi fet amb IA no vol dir fingir que l'has escrit tot tu. Vol dir poder explicar per què acceptes que entre.

Abans del commit vaig escriure una checklist ràpida:

> ## Revisió
>
> - [x] He llegit el diff. - [x] No hi ha mutació directa d'estat. - [x] Funciona amb recursos antics sense matèria. - [x] L'opció activa és comprensible. - [x] 'npm run build' passa. - [x] Prova manual feta amb tres recursos.

No la conserve sempre. No cal convertir cada canvi en oposició administrativa. Però en canvis que toquen dades, estat o interfície, eixa checklist em baixa la velocitat al punt on torne a ser responsable.

Després vaig fer el commit:

```
git commit -m "Afig ordenacio de recursos"
```

El missatge no deia "la IA ha afegit". Tampoc deia "jo he programat". Deia què havia canviat el projecte. Git no és un confessionari; és una memòria de decisions.

I tornar a entrar com a autor, al capdavall, era l'única cosa de què havia anat sempre este diari.

Apunt tècnic

Revisar codi generat per IA no és llegir cada línia com si fores compilador. És buscar zones de risc.

Mira primer el diff

```
git diff
```

Si ja has preparat els canvis:

```
git diff --cached
```

Pregunta't:

- quins arxius ha tocat?;
- havia de tocar-los tots?;
- hi ha dependències noves?;
- ha canviat textos, estils o estructura sense demanar-ho?;
- hi ha codi que no entenc prou per mantindre'l?

Zones de risc en una app web

- Estat modificat directament.
- Dades antigues que ja no encaixen amb el nou codi.
- Errors silenciosos en importació/exportació.
- Variables d'entorn exposades.
- Canvis visuals que trenquen mòbil.
- Dependències afegides per comoditat.
- Funcions grans que fan massa coses.

Prompt de revisió

> Revisa este diff com si fores una revisora estricta. Busca: - regressions possibles; - mutacions d'estat; - casos límit; - canvis fora d'abast; - riscos de seguretat o privacitat.
>
> No proposes reescriptures grans si un canvi menut resol el problema.

Criteri per acceptar

Abans de fer commit, hauries de poder respondre:

- què canvia?;

- per què calia?;
- com ho he provat?;
- què podria haver trencat?;
- com ho revertiria si falla?

La IA pot escriure el canvi. La revisió és el lloc on tu tornes a entrar com a autor.

Firmar no és reclamar el mèrit de cada línia. És acceptar que, quan eixa línia toque la vida d'algú, no podràs assenyalar la màquina i dir que tu només miraves.

Epíleg: `exit 0` — firmar el que no has escrit del tot

Al final, el tema no era `npm`.

Tampoc era React, VS Code, OpenCode, Codex, Supabase ni cap d'eixes paraules que fan que una conversa tècnica parega més important del que és.

El tema era un altre: aprendre a crear en un temps en què crear s'ha tornat més fàcil i, precisament per això, més difícil de fer amb consciència.

La IA m'ha ajudat a escriure codi que jo no hauria escrit tan ràpid. M'ha explicat errors, m'ha proposat estructures, m'ha salvat de vesprades de cerca i m'ha ficat en problemes que després he hagut d'entendre. Ha sigut accelerador, companya i, alguna vegada, font de confusió molt convincent.

Però no ha sigut autora en el sentit que importa.

No sabia per què volia guardar recursos d'aula.

No sabia quines dades eren delicades.

No sabia quan una funció era suficient.

No sabia quan una millora era una fugida cap avant.

No sabia quan dir que no.

Això em tocava a mi.

El primer dia de curs, Empar va obrir l'aplicació en l'ordinador de la seua aula sense preguntar-me res. Cap missatge, cap error, cap "com anava això?". Aquell silenci va ser la millor recensió que he tingut mai. El termini de setembre s'havia complit, i no per cap heroïcitat d'última hora: per totes les vegades que havia frenat abans d'acceptar, llegit abans de firmar i renunciat abans d'afegir.

Me'n recorde de la pregunta que vaig deixar penjada al començament d'este llibre: ara que existix, què penses fer amb ella?

Al principi em semblava una pregunta sobre una aplicació. Ara sé que era una pregunta sobre mi.

Què penses fer amb una cosa que pot créixer més ràpid que la teua comprensió?

Què penses fer amb una eina que et dona resultats abans de donar-te criteri?

Què penses fer quan una altra intel·ligència escriu amb una fluïdesa que pot fer-te oblidar que la responsabilitat continua sent teua?

La resposta no ha sigut aprendre-ho tot. Això hauria sigut una altra forma de fantasia. Ningú entén completament el món tècnic que travessa quan publica una aplicació moderna: dependències, servidors, navegadors, polítiques, claus, versions, capes que funcionen perquè moltes persones abans han fet bé la seua part. La resposta tampoc ha sigut deixar-se portar. Eixa és la fantasia contrària: creure que, com que la IA pot fer molt, jo puc mirar menys.

La resposta ha sigut més humil i més exigent: entendre prou per a respondre.

Prou per a saber què demane.

Prou per a veure quan una solució és massa gran.

Prou per a explicar on viuen les dades.

Prou per a no posar el nom d'un alumne en un lloc que no puc justificar.

Prou per a llegir un diff amb sospita honesta.

Prou per a dir "açò encara no és meu".

Una aplicació que no pots explicar encara no és teua. És un objecte trobat, per molt bé que funcione. He tingut ma casa plena d'objectes trobats: prototips brillants que no sabia defendre, botons que funcionaven sense que jo sabera per què, versions que semblaven acabades i no ho estaven. Escriure este llibre ha sigut, en bona part, la història de com alguns d'eixos objectes es van convertir en ferramentes.

I una ferramenta no és només una cosa que funciona. És una cosa que pots mantindre, limitar, entregar, retirar i explicar sense amagar-te darrere de qui t'ha ajudat a construir-la.

Crear amb IA no és deixar de ser aprenent. És descobrir que aprendre ja no és una fase prèvia a fer coses: és part del mateix acte de fer-les. Abans podia ajornar la creació fins a sentir-me preparat. Ara puc crear abans d'estar preparat, i això té una bellesa enorme. També té un perill: confondre possibilitat amb maduresa.

La IA baixa el llindar d'entrada. No baixa el llindar de responsabilitat.

De fet, potser el puja.

Perquè ara ja no puc dir tan fàcilment "no sé fer-ho" i quedar-me tranquil. Moltes coses puc començar-les. Puc demanar una interfície, una base de dades, un test, un deploy, una exportació, una migració. Puc tindre una primera versió abans que se m'acabe el café. La pregunta nova no és si puc començar.

La pregunta nova és si sé continuar sense perdre'm.

Continuar vol dir provar quan ja estàs cansat. Vol dir escriure limitacions conegudes. Vol dir no prometre "pròximament" només perquè sona millor. Vol dir revocar una clau abans de maquillar l'historial. Vol dir escoltar una petició real sense convertir-la en una reforma completa. Vol dir recordar que cada automatització toca alguna vida, encara que siga de manera menuda: una companya que prepara classe, un alumne les dades del qual no haurien d'eixir de lloc, el teu jo futur intentant entendre per què vas prendre aquella decisió.

Este llibre ha anat d'això. De frenar un poc. De mirar el diff. De fer commit. De preguntar millor. De no confondre una pantalla bonica amb una fer-

ramenta. De recordar que una aplicació no acaba quan funciona en el teu ordinador. Però també ha anat d'una cosa més ampla: de recuperar la capacitat d'aprendre davant d'un sistema massa gran per a entendre'l de colp.

Aprendre, en este context, ja no és acumular teoria abans d'actuar. És construir un diàleg responsable amb allò que encara no domines. És deixar que la IA t'acoste al problema sense deixar que et substituïsca en la mirada. És acceptar ajuda sense abdicar criteri.

El servidor de desenvolupament, aquell que s'apagava en tancar el terminal, ja no és l'única llum encesa. Ara hi ha una URL que altra gent pot obrir sense que jo estiga mirant. Això no em fa millor programador. Em fa visible d'una altra manera. El que abans era una prova privada ara és una promesa menuda davant d'algú.

Quan escric:

```
exit 0
```

no pense en una eixida perfecta. Pense en un procés que ha acabat prou bé per a deixar pas al

següent. Pense en eixa alegria rara de tancar una terminal sabent que no has entés tot el sistema, però sí les decisions que has acceptat. Pense que crear no és produir coses sense error, sinó arribar a un punt on pots respondre per elles amb honestedat.

La lliçó no és que ara sàpia fer aplicacions web.

La lliçó és que ara sé una mica millor com aprenc quan tinc al costat una intel·ligència que no es cansa i no respon per mi.

La lliçó és que una ferramenta feta amb IA no es fa meua quan la IA acaba d'escriure.

Es fa meua quan jo puc mirar-la, limitar-la, provar-la, explicar-la i, finalment, firmar-la.

No amb orgull de propietari.

Amb responsabilitat d'autor.

I això, més que una eixida, és una manera de començar.

Xuleta ràpida

Les pàgines següents no formen part del diari. Són un recull ràpid dels comandos, conceptes i hàbits que apareixen al llibre.

Comandos bàsics

```
npm install
npm run dev
npm run build
git status
git diff
git add .
git commit -m "Missatge"
```

Projecte

```
code .
```

Obri la carpeta actual en VS Code.

Git

```
git init
```

```
git status
git diff
git diff --staged
git add .
git commit -m "Canvi concret"
git tag v1.0.0
```

npm

```
npm install
npm run dev
npm run build
```

`npm run dev` arranca el servidor local. `npm run build` comprova que es pot generar la versió de producció.

Diagnòstic

Abans de dir “no funciona”, arreplega:

- comando executat;
- error complet;
- captura si és visual;
- consola del navegador;
- què esperaves;
- què ha passat.

Prompts útils

> No edites encara. Primer explica la causa probable.
>
> Proposa el canvi mínim i indica quins arxius tocaràs.
>
> Revisa este diff i busca canvis no relacionats amb la tasca.
>
> Converteix esta idea en una issue amb criteris d'acceptació.

Checklist abans de publicar

- `npm run build` passa.
- No hi ha secrets en el codi.
- `.env` està en `.gitignore`.
- Provat en mòbil.
- Provat en finestra privada.
- README actualitzat.
- Rutes revisades.
- Últim commit fet.