Claude Code + IA Local

Guía práctica para perfiles técnicos

Volumen II · Continuación de "Aprende Claude Code"

Construye herramientas de IA que se ejecutan en tu propio ordenador

y publícalas en internet

Ramón Guillamón
learntouseai@gmail.com

Edición 2026

Índice general

Cómo leer este libro

Este es el **Volumen II**. En el primer libro aprendiste *qué* es Claude Code y cómo darle tus primeras órdenes. Aquí damos el salto: vas a **construir herramientas de inteligencia artificial reales**, que se ejecutan en tu propio ordenador (tus datos no salen a la nube), y aprenderás a **publicarlas en internet** para que cualquiera las use.

No necesitas ser programador. Necesitas saber leer, tener curiosidad y estar dispuesto a copiar y pegar algún comando. Todo lo demás lo explicamos.

Puente con el Volumen I: instalación

Este libro asume que ya completaste el **Volumen I**: tienes **Claude Code instalado y funcionando** (el comando `claude` abre la herramienta en la terminal), sabes abrir la terminal y has practicado al menos un encargo sencillo. No hace falta ser experto; basta con haber seguido esos primeros pasos con calma.

Si aún no lo tienes, instálalo antes de continuar desde la documentación oficial de Anthropic: docs.anthropic.com/en/docs/claude-code. El primer capítulo de este volumen repasa la terminal, pero no sustituye la instalación inicial de Claude Code.

Qué vas a poder hacer al terminar

- Ejecutar modelos de IA **en tu propio portátil o sobremesa**, sin pagar cuotas.
- Construir aplicaciones web útiles: chatbots que responden citando documentos, lectores de PDF, asistentes de voz, simulaciones 3D para clase, y más.
- **Guardar** cada proyecto y **volver a abrirlo** cuando quieras, sin perder nada.
- **Publicar** tus aplicaciones en internet con servicios gratuitos.
- Entender el porqué de cada paso, no solo copiarlo.

"Local" en este libro: qué significa exactamente

Casi todo lo que construyas se ejecuta **en tu ordenador**: los modelos de IA (con Ollama), tus documentos y tus datos no salen a internet. Hay dos matices honestos que repetiremos cuando toque:

- **Construir** las aplicaciones se hace con Claude Code, que razona en la nube. La app resultante es local, pero durante el desarrollo no pegues datos confidenciales reales en el chat.
- Cuando **publiques** una app con IA en internet (última parte del libro), esa versión usará una IA en la nube (una *API*), porque el servidor no tiene tu Ollama. Lo verás claramente señalado.

Privacidad total al *usar* tus herramientas en tu equipo; transparencia sobre cuándo interviene la nube.

Cómo está escrito cada capítulo

Todos los capítulos siguen la misma estructura para que cojas ritmo. En cada uno encontrarás:

1. **Qué vas a construir** y **objetivos de aprendizaje**.
2. **Conceptos clave** en lenguaje llano.
3. **Antes de empezar**: lo que necesitas instalado.
4. **Paso a paso**, explicando qué hace cada comando *y por qué*.
5. **Ejecutar en tu ordenador** y **qué deberías ver**.
6. **Guardar y reabrir el proyecto**.
7. **Comprueba que funciona**, **si algo falla** y un **reto** para practicar.

Los recuadros que verás

A lo largo del libro aparecen estos avisos de colores. Aprende a reconocerlos:

Idea clave

La **idea clave** de una sección. Si solo te quedas con una cosa, que sea esta.

En cristiano: término técnico

Traducimos la jerga a lenguaje de la calle. Aquí explicaríamos, por ejemplo, qué es una "API" sin dar por hecho que ya lo sabes.

Cuidado

Un error frecuente o algo que conviene no hacer. Leer esto te ahorra tiempo y disgustos.

Comprueba que funciona

Cómo saber que lo que has hecho *de verdad* funciona: qué mensaje o resultado deberías ver.

Guardar y reabrir el proyecto

Dónde queda guardado tu trabajo y **cómo volver a abrirlo y arrancarlo otro día**. Esta sección se repite en cada capítulo: nunca perderás un proyecto.

Una convención importante

Cuando veas un bloque como este, es texto para escribir en la **terminal** (la ventana de comandos). Escribe (o pega) solo lo que va después del símbolo del sistema y pulsa *Enter*:

```
echo "¡Hola, IA local!"
```

Si no sabes qué es la terminal, no te preocupes: el primer capítulo del Volumen I lo explica, y aquí lo repasamos en cada paso.

Cuando estés listo, pasa al primer proyecto.

Parte I

Fundamentos

Capítulo 1

La terminal sin miedo (qué es un CLI)

Qué vas a aprender

Antes de construir nada, vamos a hacer las paces con una ventana que asusta a mucha gente: la **terminal**. Es la herramienta que usarás en todos los capítulos, y en diez minutos verás que no muerde.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué es un CLI y por qué los profesionales lo prefieren para muchas tareas.
- Abrir la terminal en Mac, Windows y Linux.
- Los cuatro comandos que necesitas para moverte por tus carpetas.
- Cómo no romper nada.

1.1 Qué es un CLI

CLI son las siglas de *Command Line Interface*: "interfaz de línea de comandos". Es una forma de usar el ordenador **escribiendo órdenes** en lugar de haciendo clic con el ratón.

En cristiano: CLI vs. lo de siempre

Lo que usas a diario (iconos, ventanas, ratón) es una *interfaz gráfica*. El CLI es su hermano de texto: en vez de arrastrar un archivo a una carpeta, escribes una orden que dice "mueve este archivo a esa carpeta". Parece más rústico, pero es **más rápido, más preciso y automatizable**. Y es el idioma nativo de herramientas como Claude Code.

Idea clave

No tienes que memorizar comandos. En este libro cada vez que haga falta uno, te lo damos escrito para copiar y pegar, y te explicamos qué hace.

1.2 Abrir la terminal

- **Mac**: pulsa `Cmd + Espacio`, escribe "Terminal" y pulsa Enter.
- **Windows**: menú Inicio, escribe "Terminal" (o "PowerShell") y ábrela.
- **Linux**: normalmente `Ctrl + Alt + T`.

Verás una ventana con texto y un cursor parpadeando. Eso es el *símbolo del sistema*: está esperando tus órdenes.

1.3 Los cuatro comandos para moverte

Con estos cuatro te apañas para casi todo lo del libro:

```
pwd     # ¿en qué carpeta estoy? (print working directory)
ls      # ¿qué hay aquí? (list)
cd carpeta   # entra en "carpeta" (change directory)
cd ..   # sube a la carpeta de arriba
```

En cristiano: el # y lo que va después

Todo lo que va tras una almohadilla `#` es un *comentario*: una nota para ti, no una orden. No hace falta que lo escribas.

Prueba esta secuencia sin miedo (no cambia nada, solo mira):

```
pwd
ls
cd ..
ls
```

Comprueba que funciona

Si al escribir `ls` ves una lista de nombres (Documentos, Descargas, etc.), lo estás haciendo bien: te estás moviendo por tus carpetas desde la terminal.

1.4 Cómo no romper nada

Cuidado

Reglas de oro para principiantes:

- `pwd`, `ls` y `cd` **solo miran o se mueven**: no borran nada. Úsalos con total tranquilidad.
- Desconfía de comandos que empiecen por `rm` (borrar) o que lleven `sudo` si no entiendes qué hacen. En este libro te avisamos siempre.
- Si te pierdes, cierra la ventana y abre otra: no pasa nada.

Guardar y reabrir el proyecto

La terminal no "guarda" nada por sí misma: es una ventana de mando. Lo que se guarda son las **carpetas y archivos** de tu ordenador, que siguen ahí aunque cierres la terminal. En el próximo capítulo aprendes a organizar y proteger tus proyectos para no perder trabajo nunca.

1.5 Reto para practicar

Abre la terminal, usa `cd` para llegar hasta tu carpeta de Descargas y escribe `ls` para ver qué hay dentro. Cuando lo consigas, ya sabes moverte: estás listo para el resto del libro.

Capítulo 2

Cómo trabajar con tus proyectos

Qué vas a aprender

Este capítulo es la columna vertebral del libro. Cada aplicación que construyas vivirá en una **carpeta de proyecto**. Aquí aprendes a crearla, guardarla, cerrarla y **volver a abrirla otro día** sin perder nada. Todos los demás capítulos se apoyan en lo que ves aquí.

Objetivos de aprendizaje

Al terminar sabrás:

- Organizar tus proyectos en carpetas ordenadas.
- Arrancar y detener una aplicación web.
- Reabrir un proyecto días después y continuar donde lo dejaste.
- Usar Git como una "máquina del tiempo" para no perder trabajo.

2.1 Un sitio para todo: la carpeta de proyectos

Crea una carpeta donde vivirán todos tus experimentos. Solo hay que hacerlo una vez:

```
cd ~            # ir a tu carpeta personal
mkdir proyectos-ia
cd proyectos-ia
```

En cristiano: el símbolo ~

La virgulilla `~` es un atajo que significa "mi carpeta personal de usuario". Escribir `cd ~` te lleva a casa desde cualquier sitio. Muy útil cuando te pierdes.

A partir de ahora, cada proyecto del libro será una subcarpeta aquí dentro. Así siempre sabes dónde está todo.

2.2 El ciclo de vida de un proyecto

Todos los proyectos siguen el mismo ritmo:

1. **Crear** la carpeta y entrar: `mkdir nombre` y `cd nombre`.
2. **Construir** con Claude Code (arrancándolo con `claude`).

3. **Instalar** sus piezas una vez: `npm install`.
4. **Arrancar** para probar: `npm run dev`.
5. **Detener** cuando acabas: `Ctrl + C` en esa terminal.

En cristiano: `Ctrl + C`

Es la forma universal de decirle a un programa en la terminal "para ya". No borra nada: solo apaga la aplicación que estaba corriendo. Puedes volver a arrancarla cuando quieras.

2.3 Reabrir un proyecto otro día

Esta es la parte que más tranquiliza: **nada se pierde al apagar el ordenador**. Para retomar un proyecto:

```
cd ~/proyectos-ia/nombre-del-proyecto
npm run dev
```

Y ya está. El `npm install` *no* se repite (salvo que Claude Code añada piezas nuevas). Abre la dirección local que aparezca (por ejemplo `http://localhost:3000`) y seguirás donde lo dejaste.

Comprueba que funciona

¿Dudas de dónde está un proyecto? Ve a la carpeta y escribe `ls`: si ves archivos como `package.json` o una carpeta `app`, estás en un proyecto. Si ves `README`, ábrelo: contiene las instrucciones que Claude Code escribió para ti.

2.4 Git: la máquina del tiempo de tus archivos

Git guarda "fotos" (llamadas *commits*) del estado de tu proyecto. Si algo se rompe, vuelves a una foto anterior. Es la mejor red de seguridad que existe.

En cristiano: commit

Un *commit* es una foto guardada de todos tus archivos en un momento dado, con una nota que explica qué cambió. Puedes tener cientos y saltar entre ellos. Es imposible perder trabajo si haces commits a menudo.

No hace falta que memorices comandos: pídeselo a Claude Code. Al empezar un proyecto:

```
Inicializa git en este proyecto y haz un primer commit
con todo el código. A partir de ahora, cada vez que
terminemos algo importante, recuérdame hacer un commit.
```

Y cuando quieras guardar un avance:

```
Haz un commit con los cambios de ahora y ponle un
mensaje que describa lo que hemos hecho.
```

Idea clave

Regla práctica: haz un commit cada vez que algo *funcione*. Así, si el siguiente cambio lo estropea, siempre puedes volver a la última versión que iba bien.

Guardar y reabrir el proyecto

Resumen para no perder nada nunca:

- Tu trabajo **es la carpeta** del proyecto. No la borres.
- Para cerrar: `Ctrl + C` y cierra la ventana.
- Para volver: `cd` a la carpeta y `npm run dev`.
- Para dormir tranquilo: haz **commits** de Git a menudo.
- Extra: guarda de vez en cuando una copia de la carpeta en un disco externo o en la nube.

2.5 Reto para practicar

Crea la carpeta `proyectos-ia`, entra en ella, crea dentro una carpeta `prueba`, entra, y pídele a Claude Code que inicialice Git y haga el primer commit. Ya tienes tu método de trabajo montado para todo el libro.

Capítulo 3

Escribir buenos encargos (prompts)

Qué vas a aprender

En este libro no programas: **le encargas** a Claude Code lo que quieres. La calidad de lo que recibes depende, sobre todo, de cómo escribes ese encargo. Este capítulo te enseña a pedir bien para obtener resultados que funcionan a la primera.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué es un *prompt* y por qué es la herramienta más importante del libro.
- La receta de cuatro partes de un buen encargo.
- Cómo corregir el rumbo cuando el resultado no es el que esperabas.

3.1 Qué es un prompt

En cristiano: prompt (encargo)

Un *prompt* es simplemente el mensaje que le escribes a la IA. No es magia ni un lenguaje secreto: es una instrucción en tu idioma. La diferencia entre un mal resultado y uno excelente suele estar en unas pocas frases mejor escritas.

Idea clave

Piensa en Claude Code como en un ayudante muy capaz pero **recién llegado**: hace exactamente lo que le pides, pero no adivina lo que tienes en la cabeza. Cuanto más claro y concreto seas, mejor trabaja.

3.2 La receta de un buen encargo

Un encargo sólido tiene cuatro partes. Las has visto ya en cada capítulo:

1. **Qué** quieres construir (el objetivo).
   "Crea una app web de chat con mis PDF."
2. **Con qué** herramientas (el contexto).
   "Usa Ollama con qwen3:4b y nomic-embed-text."
3. **Cómo** debe comportarse (los detalles que importan).
   "Debe citar el archivo y la página; interfaz mínima."

4. **Qué esperas de vuelta** (el formato).
   "Hazlo paso a paso y escríbeme un README."

Compara dos encargos

Flojo: "hazme un chatbot".
Bueno: "Crea una app web local de chat que responda sobre los PDF de la carpeta `docs/`, usando Ollama con qwen3:4b, citando la página de cada respuesta, con una interfaz sencilla y un README que explique cómo arrancarla".
El segundo produce algo utilizable; el primero, adivinanzas.

3.3 Trucos que marcan la diferencia

- **Pide que lo haga paso a paso** y que te explique lo que crea. Aprendes y controlas.
- **Da ejemplos** de lo que quieres ("que la factura tenga este aspecto...").
- **Fija límites**: "no uses servicios de pago", "que funcione sin conexión".
- **Pide una cosa cada vez**. Es mejor construir por partes que soltar un encargo gigante.
- **Pídele que pregunte** si algo no está claro: "si te falta información, pregúntame antes de empezar".

3.4 Cuando el resultado no es el que querías

No pasa nada: se corrige hablando. En vez de empezar de cero, **ajusta**:

```
No es lo que buscaba: el botón debería estar arriba y en
azul. Cámbialo y deja el resto igual.
```

```
Me sale este error al arrancar: [pega aquí el error tal cual].
¿Qué es y cómo lo arreglo?
```

Idea clave

Corregir es parte normal del proceso, no un fallo tuyo. Los mejores resultados salen de una conversación de ida y vuelta, no de un único encargo perfecto.

Comprueba que funciona

Un buen encargo se nota: si Claude Code empieza a construir lo que tenías en mente sin que tengas que repetírselo tres veces, lo escribiste bien.

Guardar y reabrir el proyecto

Guarda tus mejores encargos en un archivo de notas (por ejemplo `mis-prompts.txt`). Reutilizarlos y adaptarlos te ahorra tiempo en cada proyecto nuevo. Los prompts buenos son, en la práctica, tus plantillas de trabajo.

3.5 Reto para practicar

Coge cualquier proyecto de este libro y, antes de mirar el encargo que proponemos, **escribe tú el tuyo** con la receta de cuatro partes. Luego compáralos: verás qué detalles añadir la próxima vez.

Capítulo 4

IA local: elige el modelo para tu máquina

Qué vas a aprender

Vas a poner un "cerebro" de IA a funcionar en tu propio ordenador y a elegir el adecuado según tu equipo. Este capítulo es la base de casi todos los proyectos del libro.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué programas usar para ejecutar modelos en local (Ollama y LM Studio).
- Qué es la cuantización y por qué te deja usar modelos grandes en equipos modestos.
- Qué modelo elegir según tu portátil, tu GPU o tu Mac.

4.1 Dos programas para ejecutar IA en local

- **Ollama** — gratuito, se maneja con comandos sencillos. Ideal para conectar modelos a tus aplicaciones. Es el que usamos en el libro por defecto. ollama.com
- **LM Studio** — aplicación con ventana gráfica para descargar y chatear con modelos sin tocar la terminal. Perfecto para probar y comparar. lmstudio.ai

En cristiano: ¿cuál elijo?

Usa **LM Studio** para trastear y ver qué modelo te gusta (todo con el ratón). Usa **Ollama** cuando quieras que tus aplicaciones hablen con el modelo automáticamente. En la práctica muchos tienen los dos.

4.2 Cuantización: modelos grandes en equipos pequeños

Un modelo "en crudo" puede ocupar muchísima memoria. La **cuantización** lo comprime para que quepa en tu equipo perdiendo muy poca calidad.

En cristiano: cuantización (los `Q4`, `Q8`...)

Es como pasar una foto RAW enorme a un JPG: ocupa mucho menos y a simple vista se ve casi igual. `Q4` comprime bastante (rápido, poca memoria); `Q8` comprime menos (más fiel, más pesado). Para empezar, `Q4` es una gran relación calidad/tamaño.

4.3 Qué modelo elegir (edición 2026)

Los modelos evolucionan rápido; estas familias son las recomendables a fecha de 2026. Elige por la memoria de tu equipo:

Tu equipo	Modelos recomendados (empieza por el primero)
Portátil 8 GB RAM	Qwen3.5 (2B–4B), Gemma 4 pequeño, Llama 3.2 (1B–3B), Phi-4-mini
Portátil 16 GB RAM	Qwen3.5 4B, Gemma 4 mediano, Ministral 3 (8B), Phi-4-mini
GPU RTX 8–12 GB	Qwen3.5 9B, Gemma 4 (Q4), Llama 3.1 8B, phi-4 (14B, Q4)
GPU RTX 16–24 GB	Qwen3.6 (27B / 35B MoE), Phi-4-reasoning, Gemma 4 grande

Idea clave

Regla sencilla: **empieza pequeño**. Un modelo de 4B que responde al instante es más útil para aprender que uno enorme que va a trompicones. Cuando domines el flujo, sube de tamaño y compara.

En cristiano: ¿y un PC sin GPU potente, o un Mac?

Los **Mac con chip M** (Apple Silicon) ejecutan modelos sorprendentemente bien gracias a su memoria unificada; Ollama los aprovecha automáticamente. En un **PC con tarjeta NVIDIA RTX**, el modelo corre en la GPU y vuela. Y equipos nuevos tipo **NVIDIA DGX Spark** están pensados justo para esto. Sin GPU, funciona igual pero más despacio: usa modelos pequeños.

4.4 Pruébalo ahora

Descarga un modelo y háblale, sin escribir código:

```
ollama pull qwen3:4b
ollama run qwen3:4b "Explícame qué es la energía solar en dos frases"
```

Comprueba que funciona

Si te responde en tu terminal con un par de frases coherentes, **ya tienes inteligencia artificial corriendo en tu ordenador**, gratis y sin conexión. Escribe `/bye` para salir del chat.

Guardar y reabrir el proyecto

Los modelos que descargas con `ollama pull` se guardan una sola vez en tu ordenador y quedan disponibles para todos tus proyectos. Para ver los que tienes: `ollama list`. Para liberar espacio y borrar uno: `ollama rm nombre-del-modelo`.

4.5 Reto para practicar

Descarga dos modelos de distinto tamaño (por ejemplo `qwen3:4b` y un Gemma). Hazles la misma pregunta con `ollama run` y compara la calidad y la velocidad. Así aprendes a elegir el equilibrio que te conviene.

Capítulo 5

Cuando algo se rompe: depurar y proteger tu trabajo

Qué vas a aprender

Tarde o temprano algo fallará: un error al arrancar, una instalación que se atasca, Claude Code que hace algo distinto a lo que querías. No es un drama: es parte de construir. Este capítulo te da un método sereno para resolverlo y, sobre todo, para **no perder nunca tu trabajo** ni exponer tus datos.

Objetivos de aprendizaje

Al terminar sabrás:

- Leer un error sin asustarte y resolverlo con Claude Code.
- Un método de depuración paso a paso que sirve para todo.
- Proteger tus claves, tus datos personales y tus copias de seguridad.

5.1 Los errores son mensajes, no castigos

En cristiano: qué es un "error" en la terminal

Cuando algo va mal, el ordenador escribe un texto largo (a veces en rojo) que parece amenazante. En realidad es una *pista*: te dice qué esperaba y qué encontró. No tienes que entenderlo tú: tu trabajo es **copiarlo entero** y pasárselo a Claude Code.

Idea clave

La frase más útil del libro: *"Me sale este error, ¿qué significa y cómo lo arreglo?"* seguida del error pegado tal cual. Depurar es una de las cosas que mejor hace Claude Code.

5.2 Método de depuración en cinco pasos

Cuando algo no funcione, ve en orden. No saltes pasos:

1. **Lee el mensaje.** La última línea suele decir lo esencial.
2. **Aísla qué cambió.** ¿Funcionaba antes? ¿Qué hiciste justo antes de que fallara?

3. **Comprueba lo básico.** ¿Está Ollama en marcha? ¿Hiciste `npm install`? ¿Estás en la carpeta correcta (`pwd`)?
4. **Pásaselo a Claude Code** con el error completo y qué estabas haciendo.
5. **Un cambio cada vez.** Aplica una solución, prueba, y solo si no va, prueba la siguiente.

Cuidado

Evita el "cambiarlo todo a la vez" con la esperanza de que algo funcione. Si tocas cinco cosas y arregla, no sabrás cuál era y volverá a pasar. Un cambio, una prueba.

5.3 La red de seguridad: Git y copias

Si tienes miedo de "romper algo", la solución es poder volver atrás. Ya lo viste en el capítulo de proyectos, pero aquí es donde de verdad te salva:

```
Algo se ha estropeado y no sé qué. Vuelve al último commit
que funcionaba y explícame qué has revertido.
```

Idea clave

Haz un commit **cada vez que algo funcione**. Así, ante cualquier destrozo, siempre hay un punto seguro al que regresar. Es la diferencia entre un susto y un desastre.

5.4 Protege tus datos y tus claves

Cuando tus proyectos manejan información real —facturas, documentos de clientes, claves de API— la seguridad deja de ser opcional.

- **Claves y contraseñas nunca en el código.** Van en un archivo `.env` que *no* se sube a internet. Pídeselo a Claude Code: *"asegúrate de que mis claves están en .env y de que .env está en el .gitignore"*.
- **No subas datos personales a GitHub.** Antes de publicar un proyecto, revisa que no haya documentos de clientes ni bases de datos reales dentro.
- **Cuidado al construir con datos sensibles.** Recuerda: la app corre en local, pero Claude Code razona en la nube. No pegues datos confidenciales reales en el chat mientras construyes; usa ejemplos ficticios.
- **Copias de seguridad de verdad.** Una copia que nunca has probado a restaurar no es una copia. De vez en cuando, comprueba que puedes recuperar tus datos.

En cristiano: el archivo `.gitignore`

Es una lista de cosas que Git **debe ignorar** y nunca subir: claves, datos, archivos temporales. Es tu primera línea de defensa contra publicar sin querer algo privado.

Comprueba que funciona

Antes de publicar cualquier proyecto en internet, hazte tres preguntas: ¿hay alguna clave en el código? ¿hay datos personales reales en las carpetas? ¿está el `.env` en el `.gitignore`? Si las tres respuestas son correctas, adelante.

Guardar y reabrir el proyecto

Rutina de seguridad que vale para todos tus proyectos:

- Commit de Git cuando algo funcione.
- Claves en `.env`, nunca en el código ni en GitHub.
- Copia de la carpeta (y de la base de datos) fuera del ordenador cada cierto tiempo.
- Prueba de vez en cuando que sabes restaurar esa copia.

5.5 Reto para practicar

Provoca un error a propósito en un proyecto de prueba (por ejemplo, borra una línea del código), observa el mensaje, pégaselo a Claude Code y arréglalo. Perder el miedo a los errores es lo que te convierte en autónomo de verdad.

Parte II

Construye tus herramientas de IA

Capítulo 6

Un chatbot que responde citando la ley

Qué vas a construir

Una aplicación web sencilla donde escribes una pregunta en lenguaje normal —por ejemplo, *"¿cuántos días de permiso por mudanza tengo?"*— y un asistente te responde **basándose en documentos legales que tú le has dado** (leyes, un convenio, un reglamento en PDF), **citando el fragmento** de donde saca la respuesta. Y lo mejor: **la aplicación funciona entera en tu ordenador**. Una vez construida, tus documentos no salen a internet.

En cristiano: "local" de verdad, con un matiz honesto

La *aplicación* corre 100 % en tu máquina: cuando la usas, ningún documento se envía a ningún servidor. Ahora bien, para *construirla* usamos Claude Code, que se apoya en la nube durante el desarrollo (lee y escribe archivos en tu equipo, pero razona en servidores externos). En resumen: privacidad total *al usar* el asistente; durante la *construcción*, no pegues datos confidenciales en el chat de Claude Code.

Objetivos de aprendizaje

Al terminar este capítulo sabrás:

- Qué es un modelo de lenguaje "local" y por qué te interesa para datos sensibles.
- Qué es **RAG**, la técnica que hace que la IA responda con *tus* documentos y no se los invente.
- Cómo pedirle a Claude Code que te construya la aplicación paso a paso.
- Cómo ejecutarla, guardarla y volver a abrirla otro día.

6.1 Conceptos clave (en dos minutos)

Antes de teclear nada, entendamos *qué* estamos montando. Son solo tres ideas.

6.1.1 Modelo de lenguaje local

Un "modelo de lenguaje" es el cerebro que entiende y redacta texto (lo mismo que hay detrás de un chatbot conocido). **Local** significa que ese cerebro se descarga y se ejecuta en tu propia máquina.

En cristiano: modelo local

Piensa en la diferencia entre ver una película *en streaming* (necesitas internet y una cuenta) y tenerla *descargada* en el disco duro (la ves cuando quieras, sin conexión y sin que nadie sepa qué ves). Un modelo local es la versión "descargada" de la IA: privada, gratuita de usar y siempre disponible.

Idea clave

Para un abogado, una gestoría o cualquiera que maneje datos confidenciales, lo local no es un capricho técnico: es que **los documentos de tus clientes nunca salen de tu ordenador**.

6.1.2 Por qué la IA "se inventa" cosas y cómo evitarlo

Un modelo, por sí solo, responde con lo que "recuerda" de su entrenamiento. A veces acierta y a veces **inventa con total seguridad** (a esto se le llama *alucinación*). Para un tema legal, eso es inaceptable.

Cuidado

Esto es una herramienta para **buscar y redactar más rápido**, no un asesor jurídico. RAG *reduce* mucho los inventos, pero **no los elimina**, y citar un fragmento no garantiza que la interpretación sea correcta. Contrasta siempre con la norma vigente y con criterio profesional.

6.1.3 RAG: darle a la IA los documentos correctos

La solución se llama **RAG** (*Retrieval-Augmented Generation*, o "generación apoyada en búsqueda"). Funciona en dos tiempos:

1. **Buscar**: cuando preguntas algo, el sistema busca en *tus* documentos los fragmentos más relacionados con tu pregunta.
2. **Responder**: le pasa esos fragmentos al modelo y le dice: *"responde usando SOLO esto, y cita de dónde lo sacas"*.

En cristiano: RAG

Es la diferencia entre un examen "de memoria" y un examen "con apuntes". RAG le da apuntes a la IA —tus documentos— justo antes de responder. Así contesta con hechos que puedes verificar, no con lo que cree recordar.

6.2 Antes de empezar

Requisitos

Necesitas tres cosas instaladas. Si ya seguiste el Volumen I, tendrás Claude Code; las otras dos las instalamos ahora.

- **Claude Code** — para que construya la aplicación por ti.
- **Node.js** — el motor que ejecuta aplicaciones web. Descárgalo de nodejs.org (versión LTS).
- **Ollama** — el programa que descarga y ejecuta modelos de IA en local. Lo instalamos en el paso 1.

Con un portátil de 8 GB de memoria RAM es suficiente para empezar; con 16 GB irá más holgado.

Privacidad y RGPD

Si indexas expedientes, contratos o datos de clientes, actúas como **responsable del tratamiento**: necesitas **base legal** (ejecución de encargo, interés legítimo documentado o consentimiento), **informar** al interesado y aplicar medidas de seguridad. La IA local ayuda a que los documentos no salgan de tu equipo, pero **no sustituye tu responsabilidad profesional** ni el deber de **secreto profesional** del abogado: revisa siempre las respuestas, no subas más datos de los imprescindibles y no uses expedientes reales mientras construyes el proyecto con Claude Code en la nube.

6.3 Paso a paso

6.3.1 Paso 1: instala Ollama y descarga un modelo

Ollama es una aplicación gratuita. Descárgala de ollama.com e instálala como cualquier otro programa. Cuando termine, abre la **terminal** y comprueba que responde:

```
ollama --version
```

Ahora descarga un modelo. Usaremos una versión pequeña y capaz, que cabe holgada en un portátil de 8 GB. Escribe:

```
ollama pull qwen3:4b
```

En cristiano: eso del `:4b`

El `:4b` indica el "tamaño" del modelo (unos 4.000 millones de parámetros). Cuanto mayor, más listo pero más lento y más memoria necesita. Empieza por `4b`; si tu equipo es potente, más adelante puedes probar variantes mayores y notar la diferencia.

Esto descarga el modelo (unos gigas: tardará según tu conexión). Cuando acabe, también necesitamos un modelo "de embeddings", que es el que sabe *buscar* fragmentos parecidos:

```
ollama pull nomic-embed-text
```

En cristiano: embeddings

Un "embedding" convierte un trozo de texto en una lista de números que representa su *significado*. Dos textos que hablan de lo mismo tendrán números parecidos. Gracias a eso, el sistema encuentra el fragmento de la ley que *significa* lo que preguntaste, aunque no uses las mismas palabras.

Comprueba que funciona

Escribe `ollama list` y deberías ver `qwen3:4b` y `nomic-embed-text` en la lista. Eso confirma que están *descargados*. Para confirmar que Ollama *funciona*, prueba: `ollama run qwen3:4b "hola"` y verifica que te contesta.

Cuidado

Ollama tiene que estar **en marcha** para que la app funcione. En Mac y Windows, al instalarlo se abre solo y deja un icono en la barra (menús / bandeja). Si lo cerraste, ábrelo de nuevo antes de arrancar la aplicación.

6.3.2 Paso 2: crea la carpeta del proyecto

Todo proyecto vive en su propia carpeta. Vamos a crear una y entrar en ella:

```
mkdir chatbot-legal
cd chatbot-legal
```

En cristiano: carpeta y "cd"

`mkdir` crea una carpeta (*make directory*). `cd` entra en ella (*change directory*). A partir de ahora, todo lo que hagas en la terminal ocurrirá "dentro" de esa carpeta, igual que cuando haces doble clic en una carpeta del explorador de archivos.

6.3.3 Paso 3: pídele a Claude Code que construya la app

Aquí está la magia del libro: no vas a escribir el programa a mano. Vas a **describir lo que quieres** y Claude Code lo construirá. Arranca Claude Code dentro de la carpeta:

```
claude
```

Cuando se abra, pega esta petición (puedes adaptarla a tu gusto):

```
Crea una aplicación web sencilla de chat con RAG que funcione
100% en local. Requisitos:
- Usa Ollama con el modelo "qwen3:4b" para responder y
  "nomic-embed-text" para los embeddings.
- Debe leer los PDF que yo ponga en una carpeta "documentos/":
  extraer su texto, trocearlo, calcular embeddings y guardar
  ese índice en local para no recalcularlo cada vez.
- Que haya un comando para reindexar cuando añada PDF nuevos.
- Cuando responda, debe CITAR el fragmento y el archivo del
  que ha sacado la información.
- Interfaz web mínima: un cuadro de texto y las respuestas debajo.
- Explícame en un README cómo arrancarla y cómo añadir mis PDF.
Hazlo paso a paso y explícame qué archivos creas.
```

Idea clave

Fíjate en cómo está escrita la petición: dice *qué* queremos, *con qué* herramientas y *cómo* debe comportarse (¡que cite!). Un buen encargo produce un buen resultado; es la receta de cuatro partes del capítulo "Escribir buenos encargos".

Claude Code irá creando archivos y te pedirá permiso para algunas acciones. Léelas y aprueba las que entiendas. Al terminar, tendrás una aplicación funcional y un archivo `README` con las instrucciones.

6.3.4 Paso 4: añade tus documentos

Copia dentro de la carpeta `documentos/` los PDF que quieras consultar: un convenio colectivo, el Estatuto de los Trabajadores, un reglamento interno... lo que necesites. Puedes arrastrarlos ahí con el explorador de archivos.

Cuidado

Empieza con **pocos documentos** (uno o dos) para tu primera prueba. Así verificas que todo funciona antes de meterle cientos de páginas.

Idea clave

La primera vez que añades PDF, la app los **indexa**: los trocea y calcula sus embeddings. Eso puede tardar un poco y es normal —solo pasa una vez por documento—. Cada vez que añadas PDF nuevos, ejecuta el comando de reindexar que Claude Code puso en el `README`; si no, la app no "verá" los documentos recién copiados.

6.4 Ejecutar en tu ordenador

Con los documentos en su sitio, arranca la aplicación. El `README` que generó Claude Code te dirá el comando exacto; normalmente será algo así:

```
npm install
npm run dev
```

En cristiano: npm install / npm run dev

`npm install` descarga las piezas que la aplicación necesita para funcionar (se hace una sola vez). `npm run dev` enciende la aplicación en tu ordenador en "modo desarrollo", para que puedas probarla.

En la terminal verás un mensaje con una dirección local, parecido a esto:

```
  Local:   http://localhost:3000
```

Abre esa dirección en tu navegador. Escribe una pregunta sobre tus documentos y pulsa Enter.

Comprueba que funciona

La respuesta debería aparecer en pantalla **acompañada de una cita**: el trozo de texto y el nombre del PDF de donde salió. Si ves la respuesta *y* su fuente, ¡lo has conseguido! Tienes un asistente legal privado funcionando en tu máquina.

Guardar y reabrir el proyecto

Tu proyecto ya está guardado: es la carpeta `chatbot-legal` que creaste. Nada se pierde al cerrar el ordenador.

Para cerrar hoy: en la terminal donde corre la app, pulsa `Ctrl + C` para detenerla. Luego puedes cerrar la ventana.

Para volver a abrirlo otro día:

1. Abre la terminal.
2. Entra en la carpeta: `cd chatbot-legal`
   (si no la encuentras, mira antes con `cd` a tu carpeta de usuario).
3. Arranca de nuevo: `npm run dev`
   (el `npm install` *no* hace falta repetirlo).

Abre otra vez `http://localhost:3000` y seguirás justo donde lo dejaste.

Recomendado: guarda una "foto" del proyecto con Git

Para no perder nunca tu trabajo y poder deshacer cambios, pídele a Claude Code:

```
Inicializa git en este proyecto y haz un primer commit
con todo el código actual. Explícame qué has hecho.
```

En el capítulo "Cómo trabajar con tus proyectos" explicamos Git desde cero. Por ahora basta saber que es como una máquina del tiempo para tus archivos.

6.5 Si algo falla

- **"command not found: ollama"** — Ollama no está instalado o hay que reabrir la terminal. Cierra y vuelve a abrirla; si sigue, reinstala Ollama.
- **La app no responde o da un error de conexión** — Asegúrate de que Ollama está en marcha (su icono suele estar en la barra de menús). Comprueba con `ollama list`.
- **Responde muy lento** — Es normal en la primera pregunta (el modelo "se despierta"). Si tu equipo es modesto, prueba un modelo más pequeño: `ollama pull qwen3:4b` y pídele a Claude Code que lo use.
- **No cita bien o mezcla documentos** — Empieza con menos PDF y preguntas más concretas. Puedes pedirle a Claude Code que ajuste cuántos fragmentos usa por respuesta.

Idea clave

¿Bloqueado con un error? Cópialo y pégaselo a Claude Code tal cual, diciendo "me sale este error, ¿cómo lo arreglo?". Depurar es una de las cosas que mejor hace.

6.6 Reto para practicar

Ahora que funciona, mejóralo tú:

1. Pídele a Claude Code que añada un botón para **borrar la conversación**.
2. Haz que muestre **la fecha de la ley** junto a cada cita.

3. Cambia el modelo por uno más potente si tu equipo lo permite (por ejemplo un Qwen o Gemma mayor) y compara la calidad de las respuestas.

Con esto has construido tu primera herramienta de IA privada de principio a fin. En el próximo capítulo aplicamos lo aprendido a otro problema muy común: **preguntarle a tus PDF** cualquier cosa, no solo temas legales.

Capítulo 7

Pregúntale a tus PDF

Qué vas a construir

Una aplicación donde sueltas cualquier PDF —un manual, un contrato, un artículo científico, los apuntes de una asignatura— y le haces preguntas en lenguaje normal. Te responde y te dice en qué página lo ha encontrado. Es el capítulo anterior llevado a cualquier documento, no solo legal.

Objetivos de aprendizaje

Al terminar sabrás:

- Extraer y consultar el contenido de PDF con IA local.
- Pedir resúmenes, tablas y respuestas con referencia a la página.
- Reutilizar la técnica RAG que ya conoces en un caso nuevo.

7.1 Conceptos clave

Aquí aplicamos lo mismo del chatbot legal: **RAG** (buscar en tus documentos y responder con esos fragmentos). Lo nuevo es que un PDF no siempre es texto limpio.

En cristiano: PDF "de texto" vs. PDF "escaneado"

Un PDF normal lleva el texto dentro y se puede leer directamente. Un PDF *escaneado* es en realidad una foto de cada página: para leerlo hay que aplicarle **OCR** (reconocimiento óptico de caracteres), que convierte la imagen en texto. Si tu documento es un escaneo, pídele a Claude Code que añada OCR.

7.2 Antes de empezar

Requisitos

Los mismos que ya tienes: **Claude Code**, **Node.js** y **Ollama** con `qwen3:4b` y `nomic-embed-text` descargados (capítulo de IA local).

7.3 Paso a paso

Crea el proyecto y arranca Claude Code:

```
cd ~/proyectos-ia
mkdir pregunta-pdf
cd pregunta-pdf
claude
```

Pégale este encargo:

```
Crea una app web local para preguntar a mis PDF con RAG.
Requisitos:
- Ollama con "qwen3:4b" (respuestas) y "nomic-embed-text".
- Puedo subir un PDF desde el navegador o dejarlo en "docs/".
- Extrae el texto; si el PDF es escaneado, aplica OCR.
- Trocea, calcula embeddings y guarda el índice en local.
- Al responder, indica el archivo y la página de la cita.
- Botones para: resumir el documento y extraer sus tablas.
- README con instrucciones de arranque y reindexado.
```

Idea clave

Un mismo patrón —RAG— resuelve muchísimos problemas: consultar leyes, manuales, historiales, documentación técnica... Domínalo una vez y lo reutilizas toda la vida.

7.4 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Abre la dirección local, sube un PDF y pregunta algo concreto que sepas que está en el documento.

Comprueba que funciona

Sube un PDF corto, pregúntale por un dato que contenga y comprueba que la respuesta **cita la página correcta**. Prueba también el botón de resumen.

Cuidado

Si responde “no encuentro nada” con un PDF que sí tiene la información, casi siempre es que **era un escaneo sin OCR** o que **no reindexaste** tras subirlo. Revisa esas dos cosas primero.

Guardar y reabrir el proyecto

Tu proyecto es la carpeta `pregunta-pdf`. Para cerrarlo: `Ctrl + C`. Para volver otro día: `cd ~/proyectos-ia/pregunta-pdf` y `npm run dev`. Recuerda hacer un commit de Git cuando funcione (capítulo de proyectos).

7.5 Una prueba guiada de principio a fin

Para comprobar que todo funciona sin depender de tus propios archivos, usa un PDF público del BOE. Descarga la Constitución Española (dominio público) y colócala en la carpeta `docs/` de tu proyecto:

```
mkdir -p docs
curl -L -o docs/constitucion.pdf \
  "https://www.boe.es/buscar/pdf/1978/BOE-A-1978-31229-consolidado.pdf"
npm run dev
```

Abre la app en el navegador, sube (o reindexa) `constitucion.pdf` y escribe esta pregunta exacta:

> ¿Qué establece el artículo 14 de la Constitución Española sobre la igualdad ante la ley?

Qué deberías ver: una respuesta que cite el **artículo 14** y mencione que los españoles son **iguales ante la ley**, sin discriminación por nacimiento, raza, sexo, religión, opinión u otra condición personal o social. La app debe indicar el **archivo** (`constitucion.pdf`) y una **página** donde aparece ese artículo. Si responde con contenido inventado o sin cita, reindexa el PDF y vuelve a preguntar.

Comprueba que funciona

Si la respuesta reproduce la idea del artículo 14 y señala página y archivo correctos, tu lector de PDF está funcionando de verdad.

7.6 Si algo falla

- **Texto vacío al indexar** — PDF escaneado: activa OCR.
- **Respuestas lentas** — normal en documentos largos; prueba un modelo más pequeño o reduce cuántos fragmentos usa por respuesta.
- **Cita la página equivocada** — pide a Claude Code trozos más pequeños al indexar.

7.7 Reto para practicar

Pídele a Claude Code que añada un modo “compara dos PDF” (por ejemplo dos versiones de un contrato) y que te señale las diferencias importantes.

Capítulo 8

Un chatbot que te escucha y te habla

Qué vas a construir

Un asistente al que le **hablas** por el micrófono y te **responde en voz alta**. Todo con IA local: reconocimiento de voz, cerebro y voz sintética, sin enviar tu audio a ningún servidor. Ideal para accesibilidad, atención al cliente o manos libres.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué son STT (voz a texto) y TTS (texto a voz).
- Qué herramientas open source usar en 2026 y por qué.
- Montar un asistente de voz completo en local.

8.1 Conceptos clave

Un asistente de voz encadena tres piezas:

1. **STT** (*Speech To Text*): convierte tu voz en texto.
2. El **modelo de lenguaje** (Ollama) piensa la respuesta.
3. **TTS** (*Text To Speech*): convierte la respuesta en voz.

En cristiano: STT y TTS

STT es "el oído": escucha y escribe lo que dices. TTS es "la boca": lee un texto en voz alta. Entre medias, el modelo de lenguaje es "el cerebro". Tres piezas, un asistente.

Como son tres piezas en cadena, el tiempo total de respuesta es la **suma** de las tres: lo que tarda en entenderte, en pensar y en hablar. Por eso, para que la conversación sea fluida, conviene que cada pieza sea rápida.

En cristiano: latencia y "tiempo real"

La *latencia* es lo que tardas en obtener respuesta desde que dejas de hablar. Si es de varios segundos, la charla se hace incómoda. Las herramientas "de tiempo real" (como Moonshine para el oído) empiezan a transcribir *mientras* hablas, en lugar de esperar a que termines: por eso se notan mucho más ágiles.

Idea clave

El cuello de botella casi siempre es **el cerebro** (el modelo de lenguaje), no el oído ni la boca. Si tu asistente va lento, lo primero que hay que probar es un modelo de Ollama más pequeño o más rápido.

8.2 Qué herramientas usar (edición 2026)

El panorama cambió respecto a años anteriores. Recomendaciones actuales:

Pieza	Opciones recomendadas 2026
STT (oído)	**Moonshine** (rápido, tiempo real, ideal en portátil); **Nemotron 3.5 ASR** (muchos idiomas, con GPU); *faster-whisper* para transcripción de archivos.
TTS (boca)	**Kokoro** y **piper1-gpl** (ligeros, rápidos); **MagpieTTS** (multi-idioma con GPU); **F5-TTS** si quieres clonar una voz.

Cuidado

Verás en internet guías que recomiendan *Piper* (clásico) o *XTTS*: en 2026 el primero está archivado (usa su sucesor **piper1-gpl**) y el segundo está parado. Si una guía vieja no funciona, suele ser por esto.

8.3 Antes de empezar

Requisitos

Claude Code, **Node.js**, **Ollama** (`qwen3:4b`) y un **micrófono**. Las herramientas de voz las instalará Claude Code por ti; en algún caso hará falta **Python**, que también te ayudará a instalar.

8.4 Paso a paso

```
cd ~/proyectos-ia
mkdir asistente-voz
cd asistente-voz
claude
```

```
Crea un asistente de voz que funcione 100% en local:
- STT con Moonshine (voz a texto en tiempo real).
- El cerebro es Ollama con "qwen3:4b".
- TTS con Kokoro (respuesta en voz alta), en español.
- Interfaz web: un botón de micrófono; muestra lo que
  entendió y reproduce la respuesta en audio.
- Explica en el README qué instalar y cómo arrancarlo.
Guíame paso a paso e indícame los permisos que apruebe.
```

En cristiano: permiso del micrófono

La primera vez, el navegador te pedirá permiso para usar el micrófono. Acéptalo: es una autorización local del navegador, no envía nada a internet.

Un ejemplo, paso a paso, de lo que ocurre

Para que veas la cadena completa en acción, imagina que dices *"¿qué tiempo hará mañana?"*. Por dentro sucede esto, en menos de un par de segundos:

1. El **oído** (Moonshine) escucha y escribe: `¿qué tiempo hará mañana?`
2. Ese texto va al **cerebro** (Ollama), que redacta una respuesta.
3. El texto de la respuesta va a la **boca** (Kokoro), que genera el audio.
4. El navegador reproduce ese audio: oyes la contestación.

Entender esta secuencia te ayuda a depurar: si no te entiende, el problema está en el oído; si responde raro, en el cerebro; si no suena, en la boca.

Cuidado

El eco/acople. Si los altavoces están altos, el micrófono puede "oírse a sí mismo" y el asistente se responde solo en bucle. Solución: usa auriculares, o pide a Claude Code que *silencie el micrófono mientras el asistente habla*. Es el fallo más típico y despista mucho.

8.5 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Abre la dirección local, pulsa el botón del micrófono y di algo.

Comprueba que funciona

Deberías ver escrito lo que dijiste y **oír** la respuesta. Si entiende tu voz y te contesta hablando, has montado un asistente de voz completo en tu máquina.

Guardar y reabrir el proyecto

Proyecto: carpeta `asistente-voz`. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/asistente-voz` y `npm run dev`. Si añadiste piezas de Python, no hace falta reinstalarlas cada vez.

8.6 Si algo falla

- **No capta el micrófono** — revisa el permiso del navegador y que el micro correcto esté seleccionado en el sistema.
- **La voz suena robótica o en otro idioma** — pide a Claude Code otra voz/idioma de Kokoro o prueba MagpieTTS.
- **Tarda mucho** — usa un modelo de lenguaje más pequeño; la voz en sí es rápida.

8.7 Reto para practicar

Une este capítulo con el anterior: haz que puedas **preguntarle por voz a tus PDF** y te conteste hablando. Es combinar dos proyectos que ya entiendes.

Capítulo 9

Convierte cualquier texto en audio

Qué vas a construir

Una herramienta que coge un texto —un artículo, unos apuntes, un capítulo de un libro— y lo convierte en un **archivo de audio** que puedes escuchar en el móvil o el coche. En el idioma que quieras. Perfecto para estudiar, para accesibilidad o para hacer audiolibros de tus propios materiales.

Objetivos de aprendizaje

Al terminar sabrás:

- Generar audio de calidad a partir de texto, en local y en varios idiomas.
- Producir archivos MP3 descargables.
- Elegir la voz y el idioma adecuados.

9.1 Conceptos clave

Esto es TTS (texto a voz) puro, sin micrófono ni modelo de lenguaje: solo texto que entra y audio que sale.

En cristiano: ¿por qué en local y no una web cualquiera?

Hay webs que convierten texto a voz, pero suelen tener límites, marcas de agua o cobran por textos largos, y tu texto viaja a sus servidores. En local no hay límites, es gratis y privado: puedes convertir un libro entero si quieres.

9.2 Qué herramientas usar (2026)

- **Kokoro** — ligera, rápida en CPU, buena calidad, multi-idioma. Gran punto de partida.
- **MagpieTTS** — voces de calidad de producción en 9 idiomas (incluye español); mejor con GPU.
- **F5-TTS** — si quieres *clonar* una voz concreta a partir de una muestra.

9.3 Antes de empezar

Requisitos

Claude Code y **Node.js**. Para las voces puede hacer falta **Python**; Claude Code te guía en la instalación. No necesitas Ollama en este proyecto (no hay "cerebro", solo voz).

9.4 Paso a paso

```
cd ~/proyectos-ia
mkdir texto-a-audio
cd texto-a-audio
claude
```

```
Crea una app web local de texto a voz:
- Motor Kokoro por defecto; deja preparado MagpieTTS
  como alternativa de más calidad.
- Puedo pegar texto o subir un .txt.
- Selector de idioma y de voz (incluye español).
- Botón para generar y para descargar el audio en MP3.
- Si el texto es muy largo, divídelo y únelo en un solo MP3.
- README con instrucciones.
```

Idea clave

Para textos muy largos conviene trocear y unir el audio: si no, algunos motores se atragantan. Por eso lo pedimos explícitamente en el encargo.

9.5 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Pega un párrafo, elige idioma y voz, y genera el audio.

Comprueba que funciona

Deberías poder **reproducir** el audio en la propia página y **descargar** el MP3. Pruébalo con un texto en español y con otro en inglés para ver el cambio de voz.

Guardar y reabrir el proyecto

Proyecto: carpeta `texto-a-audio`. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/texto-a-audio` y `npm run dev`. Los MP3 que generes se guardan donde tú los descargues; no dependen de que la app esté abierta.

9.6 Si algo falla

- **Voz en idioma equivocado** — selecciona la voz correcta para ese idioma; no todas hablan todos los idiomas.

- **Se corta en textos largos** — confirma que la app trocea y une; si no, pídeselo a Claude Code.
- **Suena metálica** — prueba MagpieTTS para más calidad (mejor con GPU).

9.7 Reto para practicar

Añade un "modo pódcast": que dos voces distintas lean un diálogo alternándose. Ideal para material educativo más ameno.

Capítulo 10

Simulaciones 3D para explicar en clase

Qué vas a construir

Una escena **3D interactiva** en el navegador —por ejemplo el sistema solar, una molécula o una figura geométrica que se puede girar y explorar— para usar en clases, cursos o presentaciones. Sin instalar nada raro: funciona en cualquier navegador moderno.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué son Three.js y React Three Fiber y para qué sirven.
- Crear una escena 3D interactiva describiéndosela a Claude Code.
- Publicarla para compartirla con tus alumnos (enlaza con el capítulo de publicar).

10.1 Conceptos clave

En cristiano: Three.js y React Three Fiber

Three.js es la biblioteca estándar para dibujar gráficos 3D en el navegador. **React Three Fiber** (R3F) es una forma más cómoda y ordenada de usar Three.js dentro de aplicaciones web modernas. Tú no tienes que aprender ninguna de las dos: se las describes a Claude Code y él escribe el código.

Idea clave

A fecha de 2026, Three.js (versión r185) y React Three Fiber (v9) siguen siendo el estándar para 3D en la web. Es tecnología madura y muy bien soportada: lo que construyas hoy seguirá funcionando.

10.2 Antes de empezar

Requisitos

Solo **Claude Code** y **Node.js**. Este proyecto *no* necesita IA local: es una aplicación web 3D. La IA (Claude Code) se usa para *construirla*, no para ejecutarla.

10.3 Paso a paso

```
cd ~/proyectos-ia
mkdir simulacion-3d
cd simulacion-3d
claude
```

Describe la escena que quieres. Ejemplo (adáptalo a tu asignatura):

```
Crea una web con una simulación 3D del sistema solar usando
React Three Fiber (Three.js):
- El Sol en el centro y los planetas orbitando a distintas
  velocidades.
- Se puede girar la cámara y hacer zoom con el ratón.
- Al pasar el cursor por un planeta, muestra su nombre y
  un dato curioso.
- Un control para acelerar o pausar el tiempo.
- Explícame en el README cómo cambiar datos y colores.
```

Idea clave

Cambia la escena por lo que enseñes: una célula y sus orgánulos, las capas de la Tierra, un teorema geométrico, un motor... La técnica es idéntica; solo cambia lo que describes.

10.4 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Abre la dirección local y prueba a girar la escena con el ratón.

Comprueba que funciona

Deberías ver la escena 3D y poder **rotarla y hacer zoom**. Si al pasar el ratón aparecen los nombres y datos, la interacción funciona.

Cuidado

Si la escena va a tirones, suele ser por tener demasiados elementos o texturas muy grandes. Pide a Claude Code que "optimice el rendimiento" y reduzca detalles. Un portátil normal mueve escenas sencillas sin problema.

Guardar y reabrir el proyecto

Proyecto: carpeta `simulacion-3d`. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/simulacion-3d` y `npm run dev`. Cuando esté lista, en el capítulo "Publicar en la red" aprendes a subirla para que tus alumnos la abran desde un enlace.

10.5 Si algo falla

- **Pantalla en negro** — mira si hay un error en la consola del navegador (clic derecho, "Inspeccionar"); cópiaselo a Claude Code.

- **No gira la cámara** — pide que añada los controles de órbita (“OrbitControls”).
- **Muy pesada** — reduce elementos y tamaño de texturas.

10.6 Reto para practicar

Añade botones para “visitar” cada planeta (que la cámara viaje hasta él) y un panel lateral con su ficha. Habrás convertido la simulación en una pequeña lección interactiva.

Capítulo 11

Un avatar que habla para tus cursos

Qué vas a construir

Una cara animada (un *avatar*) que **lee un texto moviendo la boca**, para presentaciones, cursos online o vídeos explicativos. Escribes lo que quieres que diga y el avatar lo narra sincronizando los labios.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué es la sincronización labial y cómo se consigue.
- Combinar voz sintética (TTS) con una cara animada.
- Generar clips para tus materiales educativos.

11.1 Conceptos clave

Este proyecto une dos cosas que ya conoces o casi: la **voz sintética** (TTS, del capítulo de audio) y una **cara animada** que mueve la boca al ritmo de esa voz.

En cristiano: sincronización labial (*lip sync*)

Es hacer que los movimientos de la boca coincidan con los sonidos. El programa analiza el audio, detecta qué sonido toca en cada instante y coloca la forma de boca correspondiente. El resultado: parece que el avatar habla de verdad.

Idea clave

No necesitas ser diseñador. Puedes empezar con un avatar 2D sencillo (una cara con unas pocas formas de boca) y queda sorprendentemente bien para cursos.

En cristiano: visemas (las "formas de boca")

Un *visema* es la forma que adopta la boca para un sonido. No hay una por letra: muchos sonidos comparten forma (la "m", la "b" y la "p" cierran los labios igual). Con **apenas 8–12 dibujos de boca** bien elegidos se cubre el habla entera. El programa decide, sonido a sonido, qué visema mostrar. Por eso un avatar convincente es más sencillo de lo que parece.

Cómo encaja con lo que ya sabes

Este proyecto es una **extensión del capítulo de texto a audio**: allí generabas la voz; aquí, además, la "pones en una cara". Si aquel te funcionó, este es el siguiente escalón natural. La novedad es solo la capa visual: los visemas sincronizados con el audio que ya sabes producir.

Requisitos

Claude Code, **Node.js** y un motor de **TTS** local (Kokoro o MagpieTTS, del capítulo de audio). Puede requerir **Python**; Claude Code te guía.

11.2 Paso a paso

```
cd ~/proyectos-ia
mkdir avatar-parlante
cd avatar-parlante
claude
```

```
Crea una app web local de "avatar que habla":
- Escribo un texto y elijo idioma/voz.
- Genera la voz con Kokoro (TTS local).
- Muestra una cara 2D sencilla que mueve la boca
  sincronizada con el audio (lip sync).
- Botón para reproducir y para exportar el resultado
  como vídeo (o como audio + animación).
- README con instrucciones y cómo cambiar el avatar.
```

Idea clave

Ejemplo resuelto. Escribe como texto de prueba: *"Hola, soy tu tutor virtual. Hoy veremos las fracciones."* Al generar, deberías oír esa frase y ver la boca abrirse en las vocales y cerrarse en la "m" de "soy" / "hoy". Si la boca se mueve pero no coincide con las palabras, es cuestión de *ajustar el desfase*, no de que esté roto (lo vemos abajo).

Cuidado

El error más común: el desfase. Es fácil que la boca vaya ligeramente adelantada o retrasada respecto a la voz. No lo tomes como un fallo grave: pide a Claude Code que "añada un pequeño ajuste (en milisegundos) para sincronizar la boca con el audio" y prueba valores hasta que cuadre. Un desfase de menos de una décima de segundo el ojo ni lo nota.

11.3 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Escribe una frase, genera y observa al avatar hablar.

Comprueba que funciona

Al reproducir, deberías **oír la voz** y ver la **boca moverse acompasada**. Si coinciden razonablemente, el lip sync funciona.

Cuidado

La sincronización perfecta es difícil; una aproximación decente ya convence en un vídeo educativo. No busques perfección de estudio: busca que se entienda y sea agradable.

Guardar y reabrir el proyecto

Proyecto: carpeta `avatar-parlante`. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/avatar-parlante` y `npm run dev`. Los vídeos que exportes se guardan como archivos independientes en tu ordenador.

11.4 Si algo falla

- **Boca desincronizada** — pide a Claude Code que ajuste el desfase entre audio y animación.
- **No exporta vídeo** — puede faltar una herramienta (por ejemplo, para unir audio e imágenes); Claude Code te dirá cuál instalar.
- **Voz robótica** — cambia de motor TTS o de voz.

11.5 Reto para practicar

Enlaza el avatar con un modelo de Ollama: que el avatar **responda** a preguntas hablando, no solo lea un texto fijo. Habrás creado un tutor virtual con cara.

Capítulo 12

Crea un tema de WordPress con IA

Qué vas a construir

Un **tema** (el diseño y la estructura visual) para WordPress, hecho a tu medida con ayuda de Claude Code. Ideal si tienes o quieres una web con WordPress y no quieres pagar una plantilla ni depender de una agencia.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué es un tema de WordPress y qué tipos hay en 2026.
- Crear un tema de bloques (el estándar actual) con Claude Code.
- Instalarlo y probarlo en un WordPress local.

12.1 Conceptos clave

En cristiano: tema (*theme*)

El tema es "la ropa" de tu web: colores, tipografías, disposición de la portada, cabecera, pie... El *contenido* (tus textos y fotos) es independiente; cambiar de tema es como cambiarle el vestido a la web sin tocar lo que dice.

Idea clave

En 2026 el estándar recomendado son los **temas de bloques** (*Full Site Editing*): permiten editar toda la web visualmente, cargan rápido y son fáciles de mantener sin saber programar. Los temas "clásicos" en PHP siguen funcionando, pero para uno nuevo, empieza con bloques.

12.2 Antes de empezar

Requisitos

Claude Code y un **WordPress donde probar**. Lo más cómodo es un WordPress local: aplicaciones gratuitas como *Local* (o *Studio*, de Automattic) instalan un WordPress en tu ordenador en un par de clics. Claude Code puede guiarte en la instalación.

12.3 Paso a paso

Crea la carpeta del tema y arranca Claude Code:

```
cd ~/proyectos-ia
mkdir tema-wordpress
cd tema-wordpress
claude
```

```
Crea un tema de bloques (Full Site Editing) para WordPress:
- Nombre del tema y un estilo limpio y moderno.
- Plantillas para portada, entradas y página.
- Colores y tipografías definidos en theme.json.
- Cabecera con menú y pie con avisos legales.
- Explícame en el README cómo instalarlo en WordPress
  y cómo cambiar colores y tipografías.
```

En cristiano: theme.json

Es el "panel de control" del tema: un único archivo donde se definen colores, tipografías y espaciados. Cambiar el aspecto de toda la web es tan fácil como editar ahí unos valores (o pedírselo a Claude Code).

12.4 Instalar y probar

El `README` te dirá cómo, pero en esencia: comprime la carpeta del tema en un `.zip` y súbelo en tu WordPress, en *Apariencia* → *Temas* → *Añadir nuevo* → *Subir*. Actívalo.

Comprueba que funciona

Tras activarlo, tu web debería mostrar el nuevo diseño. Entra en el editor visual (*Apariencia* → *Editor*) y comprueba que puedes cambiar textos y colores con el ratón.

Guardar y reabrir el proyecto

Proyecto: carpeta `tema-wordpress` (el código del tema). Guárdala con Git como cualquier proyecto. Para instalar una versión nueva en WordPress, vuelve a exportar el `.zip` y súbelo. Tu contenido de WordPress no se pierde al cambiar de tema.

12.5 Si algo falla

- **El tema no aparece al subirlo** — revisa que el `.zip` contenga los archivos en la raíz y que exista `style.css` con la cabecera del tema.
- **Se ve roto** — pega el error o una captura a Claude Code; suele ser una plantilla mal referenciada.
- **No puedo editar visualmente** — confirma que es un tema de *bloques* y que tu WordPress está actualizado.

12.6 Reto para practicar

Pide a Claude Code una variación del tema en **modo oscuro** y un patrón de sección reutilizable (por ejemplo, un bloque de “testimonios”) que puedas insertar en cualquier página.

Capítulo 13

Una web para tu servicio en minutos

Qué vas a construir

Una **página de aterrizaje** (*landing page*): una web de una sola página, atractiva y clara, para presentar tu servicio, recoger contactos o vender algo. De la idea a la web publicada, muy rápido.

Objetivos de aprendizaje

Al terminar sabrás:

- Qué debe tener una landing que convierte visitas en clientes.
- Crear una con Claude Code describiéndola en lenguaje normal.
- Prepararla para publicarla (capítulo siguiente).

13.1 Conceptos clave

En cristiano: landing page

Es la web-escaparate: lo primero que ve alguien que llega desde un anuncio, un enlace de LinkedIn o una tarjeta. Su único trabajo es que el visitante **entienda qué ofreces** y **haga una acción** (escribirte, comprar, reservar).

Idea clave

Una buena landing tiene: un titular claro (qué haces y para quién), tres beneficios, una prueba (testimonio o dato), y un botón de acción visible. Si le das esto a Claude Code, tendrás una base sólida.

13.2 Antes de empezar

Requisitos

Solo **Claude Code** y **Node.js**. No necesita IA local: es una web estática y rápida.

13.3 Paso a paso

```
cd ~/proyectos-ia
mkdir mi-landing
cd mi-landing
claude
```

```
Crea una landing page moderna y rápida para mi servicio.
Datos:
- Servicio: [describe qué ofreces y a quién].
- Titular potente y subtítulo.
- Sección de 3 beneficios con iconos.
- Un testimonio y una sección de precios (o "contáctame").
- Formulario de contacto y botón de acción destacado.
- Diseño responsive (se ve bien en móvil).
- README con instrucciones.
```

En cristiano: responsive

Que la web se *adapta* al tamaño de la pantalla: se ve bien tanto en el ordenador como en el móvil. Es imprescindible hoy: la mayoría de visitas llegan desde el teléfono.

13.4 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Abre la dirección local y revisa la página. Achica la ventana del navegador para comprobar que se ve bien en pantallas pequeñas.

Comprueba que funciona

La página debería verse profesional, con tu titular arriba y un botón de acción claro. Redúcela a tamaño móvil: los elementos deben reordenarse sin romperse.

Guardar y reabrir el proyecto

Proyecto: carpeta `mi-landing`. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/mi-landing` y `npm run dev`. En el próximo capítulo la publicas en internet gratis y con tu propio enlace.

13.5 Si algo falla

- **El formulario no envía** — una landing local no manda correos por sí sola; en el capítulo de publicar conectamos un servicio de formularios gratuito.
- **Se ve mal en móvil** — pide a Claude Code que “mejore el responsive” de la sección concreta.

13.6 Reto para practicar

Crea dos versiones del titular y pide a Claude Code que prepare la web para **probar cuál funciona mejor** (un test A/B sencillo). Aprender qué convence a tus visitantes es oro.

Capítulo 14

Un asistente de oficina para autónomos

Qué vas a construir

Una pequeña aplicación local que te ayuda con el papeleo del día a día como autónomo: **crear facturas**, llevar un registro de ingresos y gastos, y responder preguntas sobre tus propios documentos. Todo en tu ordenador, sin cuotas mensuales.

Objetivos de aprendizaje

Al terminar sabrás:

- Montar una herramienta de facturación sencilla a tu medida.
- Automatizar tareas repetitivas de oficina con IA local.
- Mantener tus datos financieros privados en tu equipo.

14.1 Conceptos clave

En cristiano: ¿por qué hacértela tú y no usar un programa de pago?

Los programas comerciales cobran cada mes y guardan tus datos en sus servidores. Una herramienta hecha a tu medida hace *exactamente* lo que necesitas, es gratis de usar y tus números no salen de tu ordenador. No sustituye a tu gestoría, pero te ahorra horas de trabajo mecánico.

Cuidado

Esto es una ayuda administrativa, **no un asesor fiscal**. Los impuestos y la normativa cambian y dependen de tu caso: confirma siempre con tu gestor o gestora antes de presentar nada.

14.2 Antes de empezar

Requisitos

Claude Code y **Node.js**. Si quieres que "lea" documentos o responda preguntas, añade **Ollama** (`qwen3:4b`). Para las facturas en PDF, Claude Code instalará lo necesario.

14.3 Paso a paso

```
cd ~/proyectos-ia
mkdir asistente-autonomo
cd asistente-autonomo
claude
```

```
Crea una app web local para autónomos:
- Crear facturas (mis datos, cliente, conceptos, importes,
  IVA e IRPF) y exportarlas en PDF con numeración automática.
- Guardar clientes y conceptos frecuentes para reutilizarlos.
- Registro de ingresos y gastos con un resumen por trimestre.
- Todo se guarda en una base de datos local en mi ordenador.
- (Opcional) un chat con Ollama para preguntar sobre mis
  facturas ("¿cuánto facturé en marzo?").
- README con instrucciones y cómo hacer copia de seguridad.
```

Idea clave

Fíjate en el detalle "copia de seguridad": cuando manejas datos que importan (facturas, clientes), pide siempre a Claude Code una forma fácil de exportar y respaldar. Es tu red de seguridad.

14.4 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Crea una factura de prueba y expórtala a PDF.

Comprueba que funciona

Deberías poder rellenar una factura, **descargarla en PDF** con los totales bien calculados (IVA e IRPF incluidos) y verla luego en el registro. Comprueba que la numeración avanza sola.

Guardar y reabrir el proyecto

Proyecto: carpeta `asistente-autonomo`. Aquí los datos importan de verdad: **haz copias de seguridad** de la base de datos con regularidad (el README te dirá qué archivo copiar) y guárdala también fuera del ordenador. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/asistente-autonomo` y `npm run dev`.

14.5 Si algo falla

- **Totales mal calculados** — dile a Claude Code el porcentaje exacto de IVA/IRPF de tu caso; que muestre el desglose.
- **Perdí datos** — por eso insistimos en las copias; restaura la última.
- **El PDF sale feo** — pide un diseño de factura más limpio con tu logo.

14.6 Reto para practicar

Añade recordatorios de **facturas pendientes de cobro** y un botón para generar el resumen trimestral listo para enviar a tu gestoría.

Capítulo 15

Una app para estudiar y aprender

Qué vas a construir

Una aplicación que convierte tus apuntes o un temario en **preguntas de test**, te examina, corrige y te explica los fallos. Sirve para preparar una oposición, un examen o cualquier materia. Con IA local: estudia sin conexión y sin coste por pregunta.

Objetivos de aprendizaje

Al terminar sabrás:

- Generar tests automáticamente a partir de tus materiales.
- Crear un sistema de estudio con corrección y explicaciones.
- Aplicar la repetición espaciada para memorizar mejor.

15.1 Conceptos clave

En cristiano: repetición espaciada

Es una técnica de estudio demostrada: repasas cada cosa *justo antes* de olvidarla, espaciando cada vez más los repasos. Lo que fallas vuelve pronto; lo que dominas, tarda en volver. Así memorizas más con menos horas.

Idea clave

La IA local es perfecta para estudiar: puede generar cientos de preguntas de tus apuntes y explicarte cada respuesta, sin límites ni suscripciones. Tú pones el temario; ella, la práctica infinita.

15.2 Antes de empezar

Requisitos

Claude Code, **Node.js** y **Ollama** (`qwen3:4b`). Ten a mano tus apuntes en texto o PDF.

15.3 Paso a paso

```
cd ~/proyectos-ia
mkdir app-estudio
cd app-estudio
claude
```

```
Crea una app web local de estudio con IA:
- Subo mis apuntes (texto o PDF) sobre un tema.
- Con Ollama ("qwen3:4b") genera preguntas tipo test
  (4 opciones) basadas SOLO en mis apuntes.
- Me examina, corrige y explica por qué falla cada opción,
  citando la parte del apunte correspondiente.
- Lleva un registro de mis aciertos y repite con
  repetición espaciada lo que fallo.
- Guarda mi progreso en local.
- README con instrucciones.
```

Cuidado

Pide siempre que las preguntas salgan **solo de tus apuntes** (es RAG, como en capítulos anteriores). Si dejas que el modelo invente, puede colar datos erróneos. Con tus materiales como fuente, estudias lo correcto.

15.4 Ejecutar en tu ordenador

```
npm install
npm run dev
```

Sube un tema corto y genera tu primer test.

Comprueba que funciona

Deberías obtener preguntas coherentes con tus apuntes, poder responderlas y recibir **corrección con explicación**. Comprueba que las preguntas falladas vuelven a aparecer más adelante.

Guardar y reabrir el proyecto

Proyecto: carpeta `app-estudio`. Tu progreso se guarda en local; haz copia de seguridad si estás preparando algo importante como una oposición. Cerrar: `Ctrl + C`. Reabrir: `cd ~/proyectos-ia/app-estudio` y `npm run dev`.

15.5 Si algo falla

- **Preguntas raras o inventadas** — refuerza que use solo tus apuntes y sube material más claro.
- **Todas muy fáciles o muy difíciles** — pide niveles de dificultad ajustables.
- **Lento al generar** — genera menos preguntas de golpe o usa un modelo algo mayor si tu equipo puede.

15.6 Reto para practicar

Añade **fichas de repaso** (*flashcards*) y un modo "examen cronometrado" que simule las condiciones reales de tu prueba.

Parte III

Sácalo al mundo

Capítulo 16

Publica tu aplicación en internet

Qué vas a aprender

Hasta ahora tus proyectos vivían en tu ordenador (`localhost`). En este capítulo aprendes a **publicarlos en internet** para que cualquiera los abra desde un enlace, con servicios gratuitos.

Objetivos de aprendizaje

Al terminar sabrás:

- La diferencia entre una app que corre en tu equipo y una publicada.
- Publicar una web con Vercel y una demo de IA con Hugging Face Spaces.
- Usar APIs gratuitas para que tu app publicada tenga IA sin tu ordenador.

16.1 Conceptos clave

En cristiano: `localhost` vs. internet

`localhost` solo existe en tu ordenador: si lo apagas, la web desaparece y nadie más la ve. **Publicar** (o *desplegar*) es copiar tu proyecto a un servidor siempre encendido, que le da una dirección pública (una URL) accesible desde cualquier parte.

Cuidado

Ojo con la IA local al publicar: tus proyectos usaban Ollama *en tu máquina*. Un servidor en la nube no tiene tu Ollama. Para la versión publicada tienes dos caminos: (1) apps **sin IA** (landings, webs, simulaciones 3D) se publican tal cual; (2) apps **con IA** deben usar una **API** en la nube (lo vemos abajo) en lugar de Ollama.

16.2 Opción A: publicar una web con Vercel

Perfecto para landings, webs y simulaciones 3D. Vercel tiene un plan gratuito (*Hobby*) para proyectos personales.

1. Sube tu proyecto a **GitHub** (pídeselo a Claude Code: "sube este proyecto a un repositorio nuevo de GitHub").
2. Entra en vercel.com, conéctate con tu cuenta de GitHub e importa el proyecto.
3. Vercel lo construye y te da una URL pública. Cada vez que actualices el código en GitHub, se republica solo.

En cristiano: GitHub

Es una web donde se guardan proyectos de código (usando Git, la "máquina del tiempo" del capítulo 2). Además de respaldar tu trabajo, sirve de puente: servicios como Vercel leen tu proyecto desde GitHub para publicarlo.

Comprueba que funciona

Abre la URL que te dio Vercel desde el móvil, con los datos móviles (sin tu wifi). Si la web carga, está **de verdad en internet**.

16.3 Opción B: una demo de IA con Hugging Face Spaces

Para enseñar una app con IA sin montar un servidor. **Hugging Face Spaces** ofrece hardware gratuito limitado (incluido *ZeroGPU*, con unos minutos de GPU gratis al día), ideal para demos y prototipos, no para uso intensivo.

Pide a Claude Code: "prepara este proyecto como un Space de Hugging Face con Gradio y explícame cómo subirlo".

16.4 APIs gratuitas: IA en la nube sin tu ordenador

Cuando publicas una app con IA, en vez de Ollama usas una **API**: un servicio en internet que ejecuta el modelo por ti. Varias tienen plan gratuito generoso para empezar:

Servicio	Bueno para
Groq	Respuestas muy rápidas; modelos abiertos tipo Llama.
Cerebras	Velocidad extrema.
SambaNova	Modelos grandes (DeepSeek, Llama 70B).
OpenRouter	Variedad; muchos modelos con opción gratuita.
Google AI Studio	Modelos Gemini con cuota gratuita.

En cristiano: API y "clave" (API key)

Una API es un enchufe: tu app se conecta al servicio y le pide respuestas. La *clave* (API key) es tu contraseña personal de ese enchufe. Trátala como una contraseña: no la publiques ni la subas a GitHub. Claude Code te enseñará a guardarla en un archivo `.env` que *no* se sube.

Cuidado

Nunca subas claves ni contraseñas a GitHub. Si Claude Code crea un archivo `.env`, comprueba que esté en el `.gitignore` (la lista de lo que Git ignora). Pídeselo explícitamente: "asegúrate de que mis claves no se suben a GitHub".

16.5 Otra vía: un VPS (servidor propio)

Si quieres que tu *propia* IA local dé servicio en internet, puedes alquilar un **VPS** (un ordenador en la nube que controlas tú) e instalar Ollama allí. Es más avanzado y suele costar unos euros al mes; para la mayoría, Vercel + una API gratuita es más que suficiente al principio.

Guardar y reabrir el proyecto

Publicar no borra tu versión local: sigues desarrollando en tu ordenador y, cuando algo esté listo, actualizas GitHub y se republica solo. Guarda la URL pública y las claves de API en un sitio seguro (un gestor de contraseñas), nunca en el código.

16.6 Reto para practicar

Publica en Vercel la landing del capítulo 11 y conéctale un formulario de contacto que te llegue por correo (hay servicios gratuitos que Claude Code sabe integrar). Tendrás tu primera web profesional en internet.

Parte IV

Anexos (avanzado)

Capítulo 17

Varios ordenadores, una sola IA

Qué vas a aprender

El capítulo más ambicioso: unir **varios ordenadores en red** para que trabajen juntos y ejecuten modelos de IA más grandes de los que cabrían en uno solo. Si tienes un par de portátiles o Macs por casa, puedes montar tu propio "mini centro de datos".

Objetivos de aprendizaje

Al terminar sabrás:

- Qué es un clúster de inferencia y cuándo merece la pena.
- Usar **exo** para repartir un modelo entre varios equipos.
- Conectar los ordenadores por red local (Ethernet o Thunderbolt).

17.1 Conceptos clave

En cristiano: clúster e "inferencia"

Inferencia es simplemente "usar" el modelo (que responda). Un *clúster* es un grupo de ordenadores que colaboran como si fueran uno. Repartir la inferencia entre varios equipos permite ejecutar modelos que no caben en la memoria de uno solo: cada máquina se encarga de una parte.

Idea clave

¿Cuándo merece la pena? Cuando quieres usar un modelo **grande** (que no entra en tu equipo) y tienes **varios ordenadores** disponibles. Para el uso normal del libro, un solo equipo basta; esto es el siguiente nivel.

17.2 exo: el clúster casero

exo (de exolabs) es un proyecto open source que une automáticamente los ordenadores de tu red y reparte el modelo entre ellos. En 2026 es una herramienta madura: detecta los equipos, equilibra la carga según la potencia de cada uno y aprovecha conexiones rápidas como Thunderbolt.

En cristiano: ¿y Ollama o LM Studio?

Ollama y LM Studio están pensados para *un* ordenador. exo es la pieza que los lleva a *varios*: coordina el conjunto. También existen otras vías (por ejemplo, el modo de red de llama.cpp), pero exo es la más sencilla para empezar en casa.

17.3 Antes de empezar

Requisitos

- **Dos o más ordenadores** (Macs con chip M y/o PCs). Cuantos más y más potentes, modelos más grandes.
- Todos en la **misma red local**. Lo ideal es conectarlos por **cable Ethernet** (o Thunderbolt entre Macs): mucho más rápido y estable que el wifi.
- **Claude Code** en uno de ellos para guiarte en la instalación de exo en cada equipo.

17.4 Paso a paso (en líneas generales)

El montaje exacto lo verás en la documentación de exo, pero la idea es esta:

1. Conecta todos los ordenadores a la misma red (mejor por cable).
2. Instala exo en **cada** equipo. Pídele a Claude Code en cada uno: "ayúdame a instalar exo y a comprobar que arranca".
3. Arranca exo en todos. Se **descubren entre sí** automáticamente y forman el clúster.
4. Desde cualquiera de ellos, lanza un modelo grande: exo lo reparte entre las máquinas y expone un punto de acceso común para tus aplicaciones.

Cuidado

La red es el cuello de botella. Con **wifi** funcionará, pero lento; con **Ethernet** o Thunderbolt notarás la diferencia. Si va a trompicones, lo primero que hay que mirar es la conexión entre equipos.

17.5 Comprueba que funciona

Comprueba que funciona

Cuando exo esté en marcha en todos los equipos, debería mostrarte cuántos nodos ha detectado y la memoria total combinada. Si ves más de un nodo y la suma de memoria de tus máquinas, **el clúster está formado**. Lanza una pregunta a un modelo grande y observa cómo responde algo que un solo equipo no podría cargar.

Guardar y reabrir el proyecto

Un clúster no es un "proyecto" con carpeta: es una configuración de tus equipos. Anota en un documento qué ordenadores lo forman, cómo los conectas y el comando para arrancar exo en cada uno, para poder reconstruirlo en minutos cuando lo necesites.

17.6 Si algo falla

- **No se ven entre ellos** — confirma que están en la misma red y que ningún cortafuegos bloquea exo.
- **Va muy lento** — pasa de wifi a cable; comprueba que ningún equipo esté saturado por otra tarea.
- **Un equipo tira del rendimiento hacia abajo** — exo reparte según capacidad, pero un nodo muy débil puede lastrar; pruébalo con y sin él.

17.7 Reto para practicar

Mide la diferencia: ejecuta el modelo más grande que quepa en **un** equipo y luego uno mayor con el **clúster**. Compara qué modelos puedes usar en cada caso. Habrás construido tu propia infraestructura de IA en casa.

Capítulo 18

Hacia dónde crece este libro

Este libro está **vivo**. La inteligencia artificial local avanza cada mes: salen modelos mejores, las técnicas de inferencia se vuelven más eficientes, la gestión de memoria mejora, aparecen nuevos frameworks y nuevas formas de afinar (*fine-tuning*) modelos con tus propios datos. Claude Code también gana funciones constantemente. Por eso esta guía se irá ampliando con nuevos capítulos y ediciones.

Si has llegado hasta aquí, ya tienes lo esencial: sabes construir herramientas de IA que se ejecutan en tu propio ordenador, entenderlas, ejecutarlas, guardarlas y volver a abrirlas. Lo demás es práctica y curiosidad.

Sugerencias y contacto

Si tienes dudas, encuentras un error o quieres proponer mejoras y nuevos capítulos, escribe a **Ramón Guillamón** — learntouseai@gmail.com. Las mejores ideas de la comunidad acaban en la siguiente edición.

Licencia

Sígueme y comparte lo que construyas

Si este libro te ha servido, cuéntamelo: me encanta ver lo que hace la comunidad.

LinkedIn	linkedin.com/in/rguillamon
X	x.com/learntouseai
GitHub	github.com/raym33
Correo	learntouseai@gmail.com

Gracias por aprender y por construir.

— Ramón Guillamón