CURSOR PARA PROGRAMAR CON IA

BIG school

Cómo utilizar esta herramienta para avanzar más rápido sin dejar de entender lo que estás construyendo.

Cursor se ha convertido en uno de los editores más populares para desarrollar con IA. Aun así, creo que hay una forma bastante mala de empezar a utilizarlo.

Abres un proyecto que apenas conoces, le pides al agente que empiece a modificar cosas y esperas a ver qué ocurre.

¿Puede hacerlo? Por supuesto. ¿Es buena idea? Para mí, no demasiado.

El problema llega después, cuando tienes diez archivos modificados delante y no sabes demasiado bien qué ha cambiado, por qué lo ha hecho así o si esa solución tenía realmente sentido.

Por eso en esta guía no quiero enseñarte todas las funciones de Cursor. Vamos a hacer algo bastante más útil: abrir un proyecto, entender cómo funciona, hacer un cambio pequeño con IA y revisar el resultado antes de aceptarlo.

1. ¿Qué es Cursor y por qué tantos developers lo están usando?

Si llevas un tiempo programando y vienes de VS Code, cuando abras Cursor seguramente te resulte bastante familiar.

Tienes el explorador de archivos, el editor, el terminal... Vamos, lo que esperas encontrar en un entorno de desarrollo moderno.

Entonces, ¿qué cambia? Pues la IA.

No me refiero simplemente a tener un chat al lado del editor para pedirle un fragmento de código. Cursor puede trabajar con el contexto de tu proyecto: buscar archivos, ayudarte a entender cómo se relacionan entre ellos y, mediante sus agentes, incluso realizar cambios sobre el código.

Y esto cambia bastante la forma en la que puedes trabajar con IA mientras programas.

A ver, piensa en cómo has utilizado la IA durante bastante tiempo.
Te encuentras con un problema, copias un fragmento de código, te vas a ChatGPT o a otra IA, explicas cómo puedes el contexto y después vuelves al editor con la respuesta.

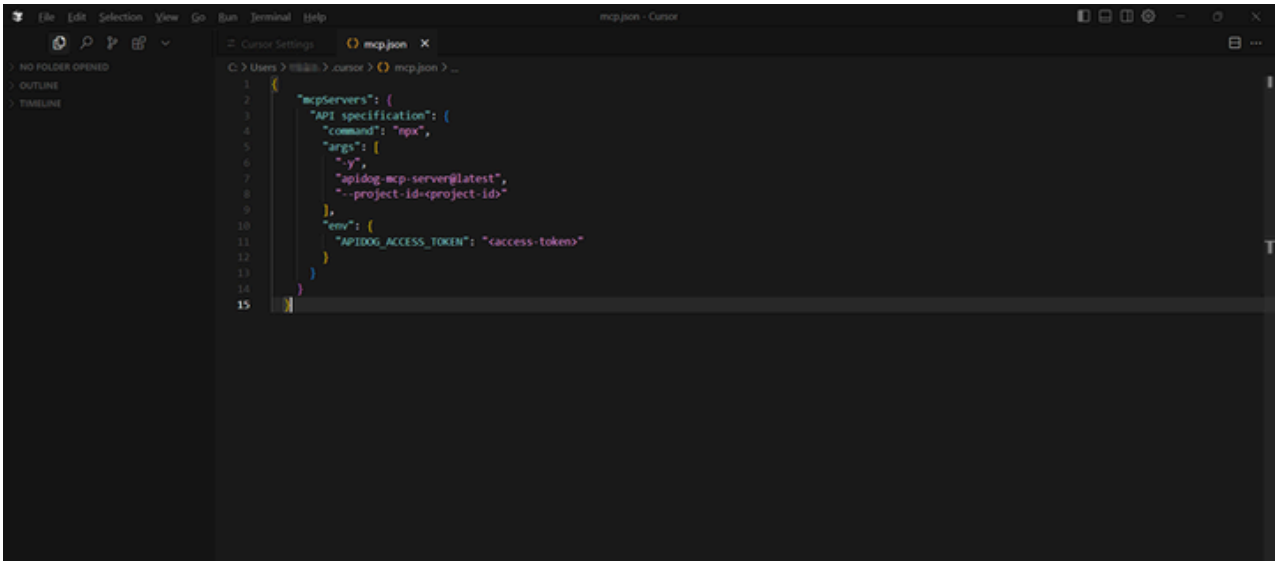
Con Cursor te ahorras buena parte de ese viaje porque la IA está trabajando dentro del mismo entorno en el que tienes el proyecto.

¿Significa esto que ya entiende perfectamente tu código y puedes dejar que haga lo que quiera? Pues no.

Y aquí quiero hacer una pausa, porque para mí esta es la parte importante de la herramienta.

Cuanto más contexto y capacidad de actuación le das a la IA, más criterio necesitas tú. Tienes que saber qué estás intentando conseguir, qué quieres que toque y cómo vas a comprobar después que lo ha hecho bien.

Por eso no quiero que empieces esta guía pidiéndole a Cursor que te programe nada. Primero vas a utilizarlo para algo mucho más útil: entender el proyecto que tienes delante.



Cursor integra el agente en el mismo entorno en el que tienes abierto tu proyecto.

2. Entiende el proyecto antes de cambiar código

Ya tienes **Cursor** abierto y el proyecto delante. Puede que lo primero que te apetezca sea pedirle que empiece a programar. Yo todavía no lo haría.

Antes aprovecharía una de las cosas más interesantes de trabajar con IA dentro del propio editor: **pedirle que te ayude a entender el proyecto antes de tocarlo.**

Esto viene especialmente bien cuando llegas a código que no has escrito tú, retomas un proyecto después de un tiempo o simplemente necesitas localizar dónde ocurre una determinada funcionalidad.

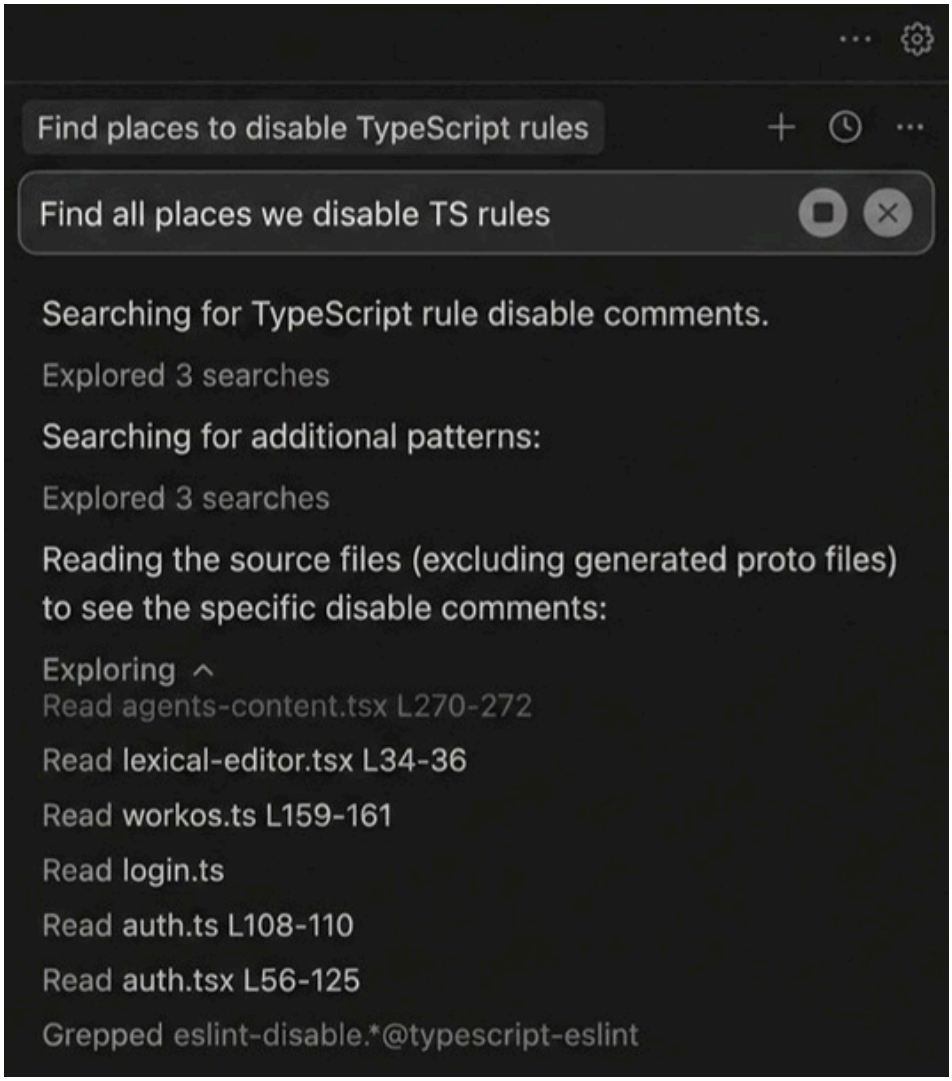
En lugar de empezar a abrir carpetas y archivos un poco a ciegas, **puedes utilizar Cursor para que investigue el proyecto y te ayude a construir ese primer mapa.**

Por ejemplo, puedes empezar pidiéndole algo así:

> *Explícame cómo está organizado este proyecto antes de modificar nada.*
>
> *Quiero entender:*
>
> 1. *Cuál es el punto de entrada.*
> 2. *Qué carpetas y módulos son los más importantes.*
> 3. *Cómo se relacionan entre ellos.*
> 4. *Dónde está la lógica principal.*
> 5. *Qué archivos debería revisar primero para entender el proyecto.*
>
> *No modifiques ningún archivo.*

Ahora, ojo con esto. La idea no es que Cursor te escriba un resumen bonito del proyecto y pases automáticamente a lo siguiente.

Si te señala tres archivos como piezas importantes, **ábrelos, léelos y comprueba que lo que te está contando tiene sentido.**



Cursor puede ahorrarte muchísimo tiempo encontrando por dónde empezar. Eso no debería convertirse en una excusa para dejar de entender lo que tienes delante. Al final, si dentro de unos minutos vas a permitir que modifique ese código, lo mínimo es que sepas qué estás tocando y por qué.

Y fíjate en que todavía no has generado ni una sola línea de código. Ya estás utilizando la IA para algo que, para mí, tiene muchísimo valor: entender mejor el proyecto antes de decidir qué cambiar.

3. Empieza por un cambio pequeño que puedas revisar

Ahora que ya tienes una idea bastante más clara de cómo está organizado el proyecto, sí puedes empezar a tocar código. Y aquí es bastante fácil venirse arriba y pedirle a Cursor que cambie media aplicación de golpe.

Yo no empezaría por ahí.

Si estás aprendiendo a trabajar con un agente dentro de tu editor, tiene mucho más sentido comenzar con una tarea pequeña cuyo resultado puedas entender y comprobar por ti mismo. No porque Cursor no pueda encargarse de algo más grande, sino porque **cuanto mayor sea el cambio, más difícil será saber si realmente ha tomado buenas decisiones.**

Por ejemplo, imagina que tienes una aplicación de tareas y quieres evitar que pueda guardarse una tarea sin título. Es un cambio pequeño: sabes exactamente qué comportamiento esperas y puedes comprobar fácilmente si funciona.

Puedes planteárselo así:

> *Quiero evitar que una tarea pueda guardarse sin título.*
>
> *Antes de modificar código, revisa cómo se crean actualmente las tareas e identifica qué archivos intervienen en ese proceso.*
>
> *El cambio debe cumplir estas condiciones:*
>
> 1. *No se puede guardar una tarea si el título está vacío.*
> 2. *Tampoco si únicamente contiene espacios.*
> 3. *Mantén la arquitectura y las tecnologías actuales.*
> 4. *No modifiques otras funcionalidades que no estén relacionadas con esta tarea.*
>
> *Antes de implementar nada, explícame brevemente cómo lo harías.*

Fíjate en que no le estás diciendo qué función tiene que escribir ni en qué archivo debe colocar cada línea. Le estás dejando investigar el proyecto y también le estás marcando con claridad el problema, los límites y qué esperas del resultado.

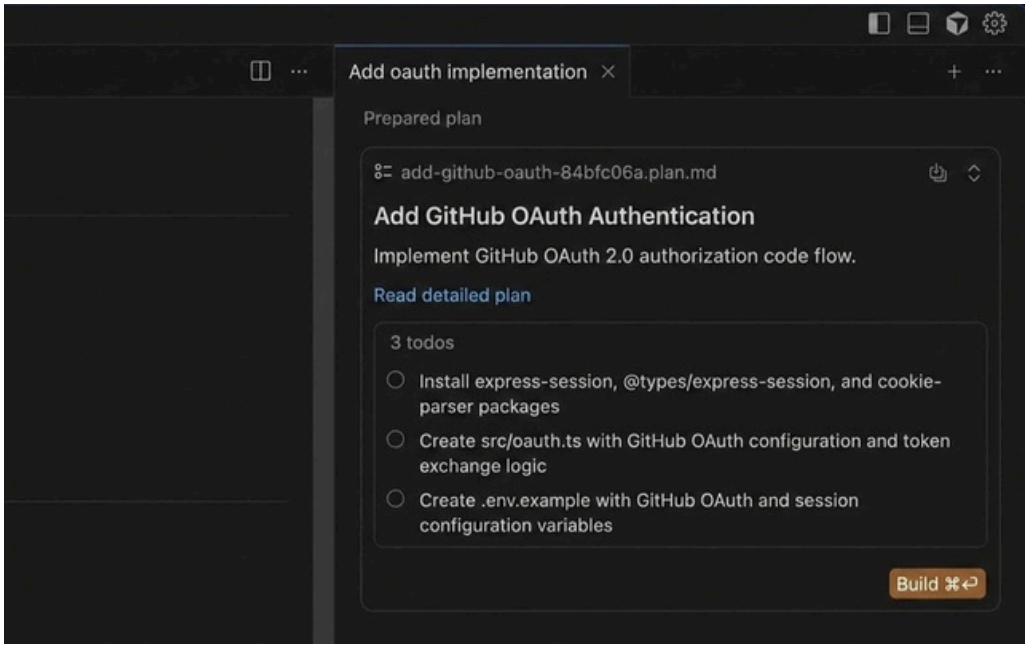
Para mí, este es un punto importante cuando trabajas con agentes. No se trata de escribir un prompt gigantesco para controlar absolutamente todo, ni de soltar un "mejora esta aplicación" y cruzar los dedos.

Cuanto más claro tengas tú el cambio que quieres conseguir, más fácil será detectar después si Cursor se ha desviado, ha añadido complejidad innecesaria o simplemente ha entendido mal lo que le estabas pidiendo.

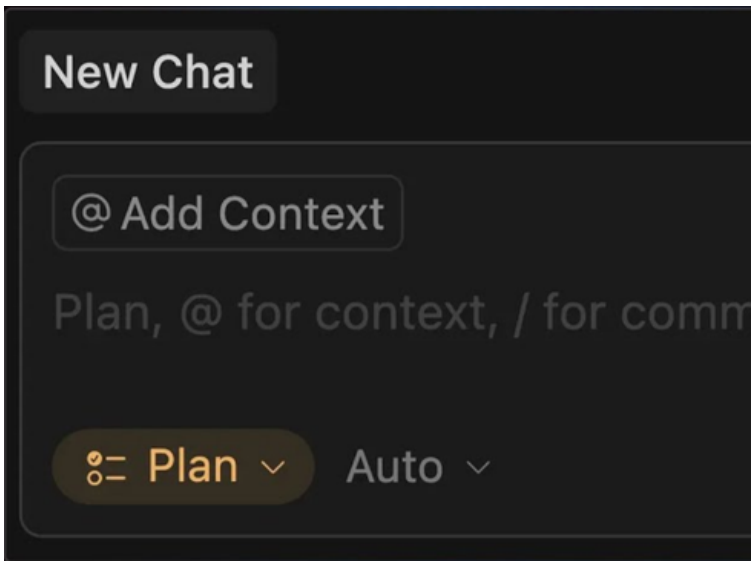
4. Antes de tocar código, revisa cómo propone resolverlo

Ya has definido un cambio pequeño y sabes qué resultado esperas. Ahora podrías dejar que Cursor empiece directamente a modificar archivos. Antes yo haría una cosa más: comprobar cómo propone resolver la tarea.

Para eso puedes utilizar Plan Mode. No necesitas activarlo para cualquier cambio mínimo. Sin embargo, cuando una tarea empieza a afectar a varias partes del proyecto o puede resolverse de distintas maneras, detenerte antes de generar código puede ahorrarte bastantes problemas después.



Activarlo es muy sencillo. Desde el campo de entrada del Agent, pulsa Shift + Tab y plantea la tarea. Cursor analizará el proyecto y, si necesita más información, te hará algunas preguntas para concretar mejor lo que quieres conseguir.



Respóndelas con calma. No son un trámite. Cuanto mejor esté definido el problema, menos decisiones tendrá que asumir el agente por su cuenta.

Después preparará un plan que puedes leer y editar antes de empezar a implementar. Aquí es donde yo me detendría un momento. Mira qué archivos quiere tocar, qué cambios propone y cómo piensa comprobar que el resultado funciona.

Si algo no encaja, corrígelo ahora. Puedes seguir hablando con el agente o editar el propio plan antes de dejar que empiece a implementar.

Antes de aprobar el plan, yo comprobaría estas cosas:

- *¿Ha entendido realmente lo que quieres conseguir?*
- *¿Va a modificar solo lo necesario?*
- *¿La solución encaja con la arquitectura actual?*
- *¿Está añadiendo complejidad o dependencias que no necesitas?*
- *¿Ha pensado cómo comprobar que el cambio funciona?*

Es mucho más barato corregir una mala decisión cuando todavía es un plan que después de tener diez archivos modificados.

Y recuerda que no necesitas montar todo este proceso para cambiar dos líneas de código. El objetivo no es añadir pasos porque sí, sino utilizar el nivel de planificación que tenga sentido para la tarea que tienes delante.

Quédate con esto: revisa el enfoque antes de dejar que el agente empiece a escribir código.

5. Deja que implemente y revisa lo que ha hecho

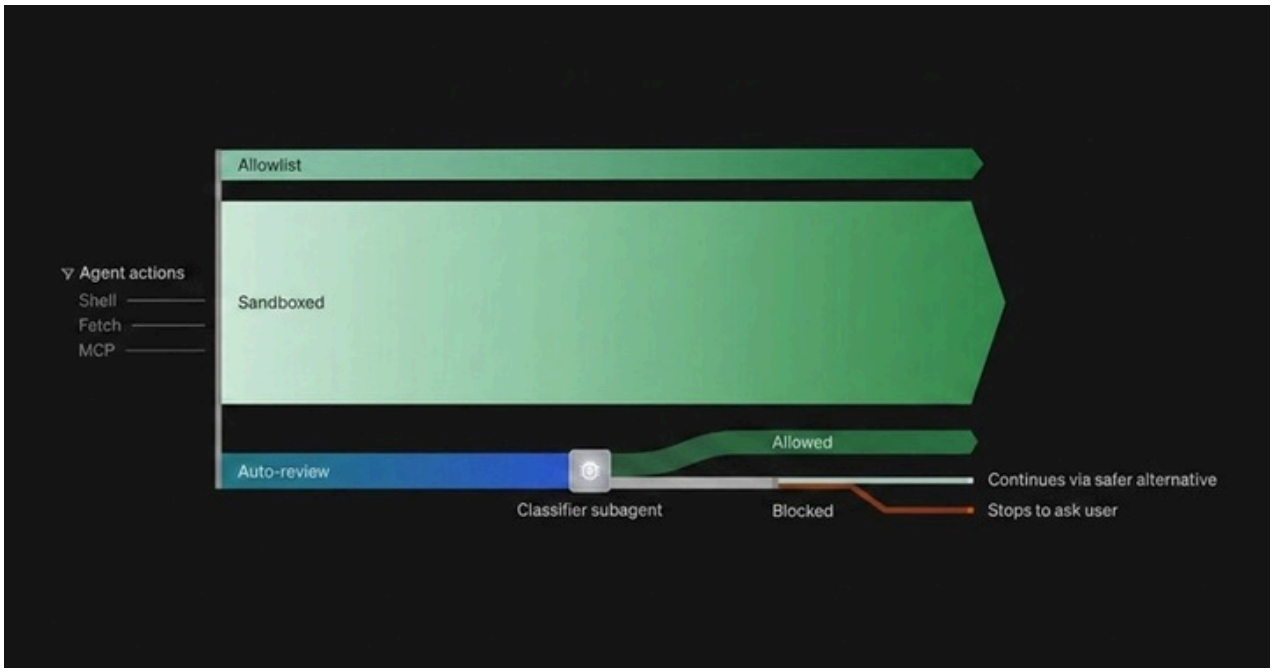
Si el plan tiene sentido, ahora sí puedes dejar que el agente implemente. Y aquí aparece otra decisión que merece la pena entender: cuánta autonomía quieres darle mientras trabaja.

Aquí Cursor también te deja decidir cuánto quieres que el agente avance por su cuenta. Auto-Review puede continuar con determinadas acciones y pedirte confirmación cuando hace falta más control.

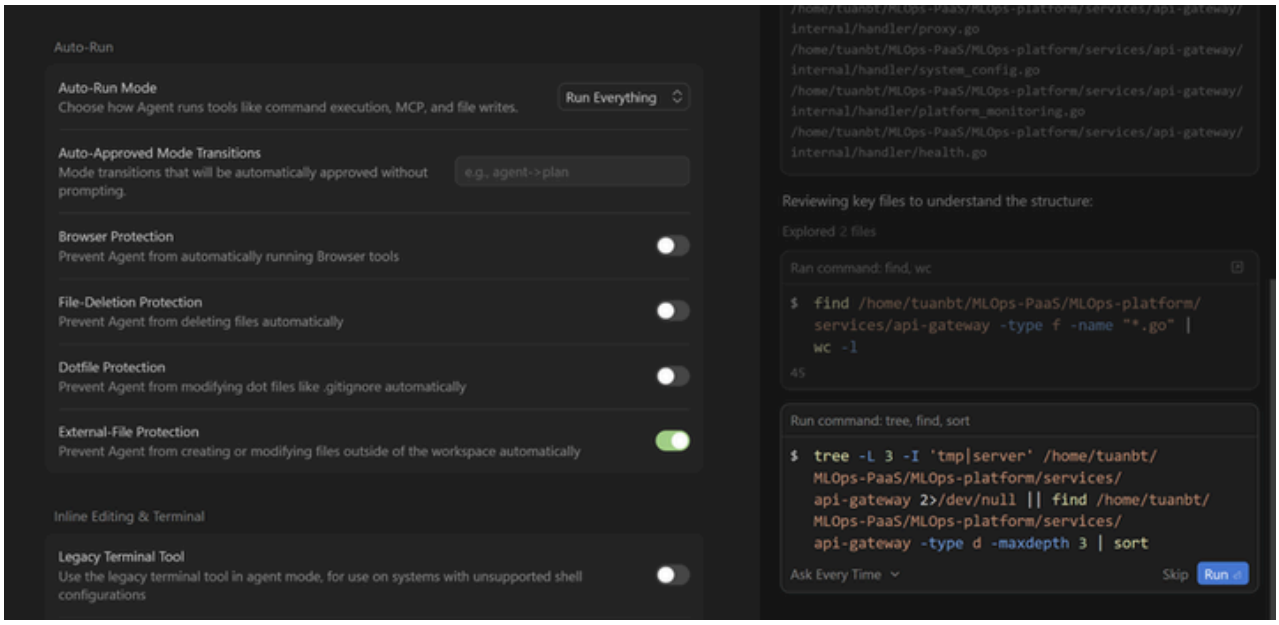
Lo importante no es aprenderte toda la configuración, sino entender que no tienes que elegir entre aprobar absolutamente todo o dejar que el agente haga cualquier cosa.

Si estás empezando con Cursor, yo mantendría los puntos de control.

No porque tengas que aprobar cada movimiento por miedo a que la IA toque algo, sino porque al principio te interesa ver qué está haciendo, qué intenta ejecutar y aprender qué puedes delegar con tranquilidad y qué merece que te pares a revisar.



A medida que conozcas mejor Cursor y tu propio flujo de trabajo, **podrás darle más autonomía donde tenga sentido**.



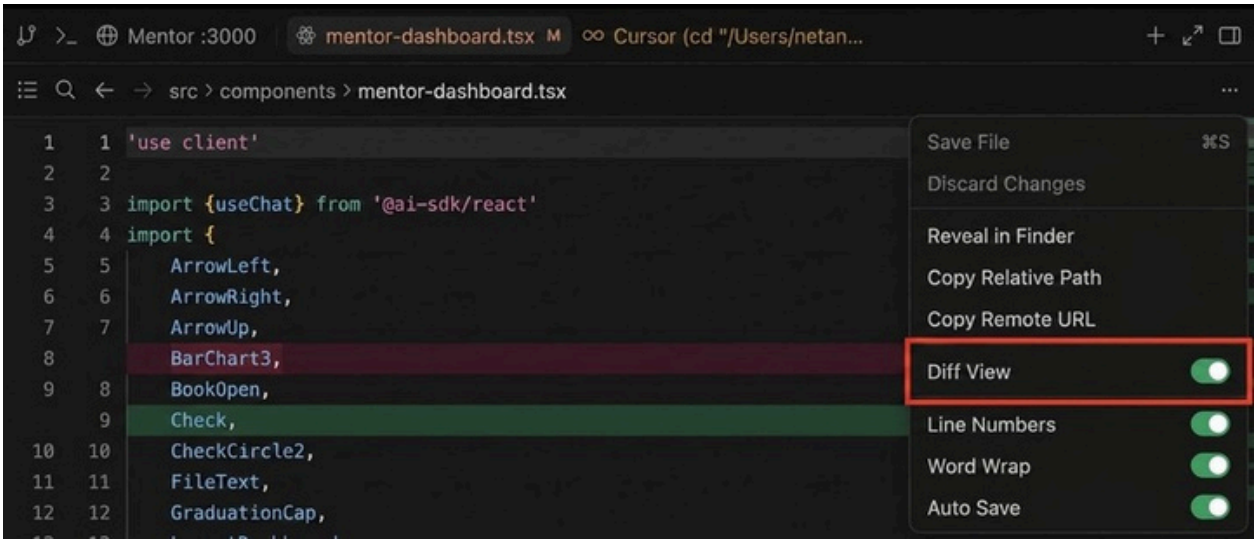
Cuando el plan tenga sentido y hayas ajustado la autonomía a lo que necesitas, deja que implemente. Cursor puede modificar archivos y ejecutar comandos. Ojo: que el agente diga que ha terminado no significa que la tarea esté terminada.

Cuando acabe, vuelve a hacer algo muy de programador: **revisar el cambio**.

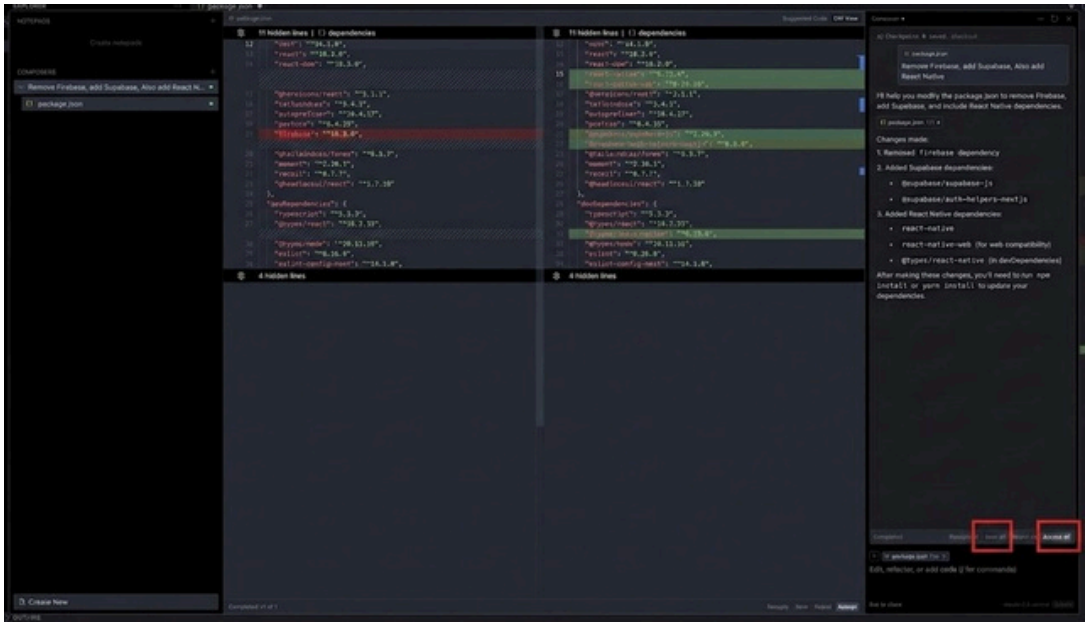
Que compile o que los tests estén en verde ayuda. Eso no sustituye una revisión. Abre **Review changes** para revisar el diff completo y comprueba si lo que ha hecho encaja de verdad con la tarea que definiste al principio.

Antes de aceptar los cambios, yo comprobaría:

- *¿Ha modificado solo los archivos que tenían sentido?*
- *¿Entiendes por qué existe cada cambio?*
- *¿Ha respetado la arquitectura y las convenciones del proyecto?*
- *¿Ha añadido código, dependencias o complejidad que no necesitabas?*
- *¿El comportamiento que definiste al principio funciona y está cubierto por tests?*



Diff View te permite comparar visualmente los cambios de un archivo. Para revisar el conjunto completo antes de aceptar el resultado, utiliza **Review changes.**



Revisa el conjunto de cambios antes de aceptar el resultado.

Si encuentras algo que no entiendes, no lo aceptes simplemente porque "parece funcionar". Pregúntale. Por ejemplo:

> *Explícame estos cambios antes de que los acepte.*
> *Quiero entender:*
> *1. Qué has modificado y por qué.*
> *2. Qué alternativas había.*
> *3. Qué riesgos tiene esta solución.*
> *4. Qué podría romperse.*
> *No realices nuevos cambios todavía.*

Cursor puede ahorrarte muchísimo tiempo escribiendo, buscando y modificando código. Perfecto. Ese ahorro no debería servir para que cada vez entiendas menos tu proyecto. **Si no puedes explicar qué ha cambiado y por qué, yo todavía no daría la tarea por terminada.**

Si quiero que te quedes con una idea de esta guía, es esta:

Deja que Cursor te quite trabajo, no criterio. Úsalo para entender mejor el proyecto, acotar el cambio, revisar el plan y validar lo que ha hecho antes de aceptarlo.

La IA puede acelerar muchísimo el trabajo. Decidir qué entra en tu código sigue siendo cosa tuya.

ACTIVA LA CAMPANITA EN YOUTUBE

PARA QUE TE AVISE CUANDO COMIENCE LA CLASE #1 DEL CURSO.

ACCEDE AQUÍ A LA PRIMERA CLASE

BIG school