GUÍA NODESTUDIO

Checklist para que tu proyecto no se rompa al crecer

Cambiá una línea sin romper cinco cosas que no tenían nada que ver.

✓ 14 puntos + guía + 1 prompt para Claude Code

1 Checklist

Marcá cada punto. Si alguno queda sin tildar, ahí hay algo que se puede romper sin que nadie se entere.

TESTS

- [ ] **Flujos críticos con tests automáticos**
  Lo que, si se rompe, deja a los usuarios sin poder usar la app o sin poder pagar tiene un test que corre solo.
- [ ] **Un test de regresión por cada bug que ya pasó**
  Cada bug arreglado deja un test que lo vigila, para que no vuelva sin avisar.
- [ ] **Los tests detectan un cambio hecho a propósito**
  Si rompés algo y todo sigue en verde, los tests no protegen nada.

DATOS Y CONTRATOS

- [ ] **Tipado estricto, sin `any` sueltos**
  Los tipos dicen qué entra y qué sale; un `any` suelto apaga esa protección justo donde se rompe.
- [ ] **Validación en los bordes**
  Formularios, endpoints y respuestas de APIs externas se validan al entrar, sin confiar en lo que llega.
- [ ] **Contratos entre frontend, backend y APIs**
  Un esquema compartido en vez de dos copias que se desincronizan.

1 Checklist (cont.)

ESTRUCTURA

- [ ] **Sin lógica de negocio duplicada**
  Cada regla vive en un solo lugar: si cambia, se cambia una vez.
- [ ] **Sin dependencias ocultas**
  Variables globales, orden de ejecución implícito, variables de entorno que nadie documentó.
- [ ] **Componentes desacoplados**
  Se puede tocar uno sin arrastrar a los demás.
- [ ] **Mapa de las partes frágiles**
  Una lista de qué partes se rompen más fácil y qué cambios las rompen.

DEPLOY

- [ ] **Migraciones seguras con rollback probado**
  Cada migración de base de datos tiene marcha atrás, y probaste que esa marcha atrás funciona.
- [ ] **CI antes de cada deploy**
  Lint, tests, build y type check corren solos antes de publicar.
- [ ] **Rama principal protegida**
  No se mergea si el CI falla.
- [ ] **Nada depende de que alguien se acuerde**
  Todo paso manual está automatizado o escrito en un checklist.

2 Guía: cómo usarlo

Cuándo

- Cuando el proyecto pasa de prototipo a tener usuarios reales.
- Antes de sumar una feature grande.
- Cuando arreglar una cosa empezó a romper otra.

Paso a paso

1. Abrí Claude Code en la carpeta del proyecto.
2. Pegá el prompt de la página siguiente.
3. **Leé el plan antes de aprobar.** Claude lo muestra ordenado por riesgo y espera tu OK.
4. Dejá que trabaje en una rama aparte.
5. Revisá el resumen final: qué agregó, qué sigue frágil y qué te queda configurar a mano (por ejemplo, proteger la rama `main` en GitHub).

Cómo leer lo que te devuelve

- **El mapa de riesgos:** qué partes son frágiles y qué cambios podrían romperlas.
- **Los tests nuevos:** qué flujos críticos y qué bugs viejos quedaron cubiertos.
- **La prueba de fuego:** cada cambio a propósito tiene que hacer fallar al menos un test. Si no falla ninguno, ahí falta un test.
- **Bugs que destapen los tests nuevos:** Claude los anota y te pregunta antes de arreglarlos.

> **TIP**
> La prueba de fuego es lo que distingue "tengo tests" de "mis tests me protegen".

Para que no se vuelva a ensuciar

Sumá esta regla al `CLAUDE.md` del proyecto:

```
Cada bug arreglado lleva su test de regresión.
No se mergea con el CI en rojo.
```

Y volvé a correr el prompt cada tanto.

3 El prompt completo

Abrí Claude Code en la carpeta del proyecto y pegá esto:

PROMPT PARA PEGAR EN CLAUDE CODE

```
Hey Claude, no quiero cambiar una línea y romper cinco cosas que no tenían nada que ver. Revisá todo este proyecto y dejalo listo para crecer sin miedo a romperlo.

Antes de tocar nada, analizalo y mostrame un plan ordenado por riesgo. Esperá mi OK.

1. Identificá los flujos críticos (lo que, si se rompe, deja a los usuarios sin poder usar la app o sin poder pagar) y asegurate de que tengan tests automáticos.
2. Agregá tests de regresión para los bugs que ya existieron. Buscalos en el historial de git y en los commits de "fix".
3. Revisá el tipado, las validaciones y los contratos entre frontend, backend y APIs.
4. Detectá código duplicado, dependencias ocultas y componentes demasiado acoplados.
5. Asegurate de que las migraciones de base de datos sean seguras y tengan una estrategia de rollback.
6. Configurá checks automáticos antes de cada deploy para que no se publique código roto: linting, tests, build y type checking.
7. Identificá las partes más frágiles del proyecto y explicá qué cambios podrían romperlas.
8. Después hacé un cambio pequeño a propósito en cada parte crítica y comprobá que los tests lo detectan. Revertí cada cambio al terminar. Si un cambio no hace fallar ningún test, escribí el test que falta.
9. Encontrá cualquier otro lugar donde el proyecto dependa de que nadie cometa un error.

Reglas:
- Trabajá en una rama nueva, nunca directo en main.
- No toques producción, datos reales ni variables de entorno reales. Las migraciones, solo en local.
- Si un test nuevo destapa un bug, anotalo y preguntame antes de arreglarlo: no cambies lo que ve el usuario sin avisar.
- Al final, resumime qué tests y checks agregaste, qué partes siguen frágiles y qué me queda configurar a mano (por ejemplo, proteger la rama main en GitHub).

Y no cometas errores.
```



NodeStudio

Automatizamos procesos de
negocio con IA y sistemas a medida.

nodestudioai.com