GUÍA PASO A PASO

Especificación para agentes de IA

La plantilla de especificación con objetivo, restricciones y criterios de aceptación, dos ejemplos llenos (corregir un bug y una función nueva), un script en Node que la revisa antes de empezar y los prompts para construir y revisar. Con la checklist.

PARA	Quien trabaja con agentes de código como Claude Code en proyectos web
PROBADO CON	Node 22.22.3 · check-spec.mjs sobre la plantilla y los dos ejemplos de esta guía
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/best-practices
ARTÍCULO	ecomlabs.dev/g/spec
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Las cuatro secciones
Paso 01	La plantilla en blanco
Paso 02	Ejemplo lleno: corregir un bug
Paso 03	Ejemplo lleno: una función nueva
Paso 04	Revisar, construir y revisar otra vez
Al terminar	Checklist

ANTES DE EMPEZAR

Las cuatro secciones

- **Objetivo:** qué cambia, dónde y para quién. Describe el resultado, no la implementación.
- **Restricciones:** lo que el agente no puede leer en el repo: qué archivos toca, qué utilidades usa, qué dependencias no entran y tu preferencia cuando hay dos caminos válidos.
- **Criterios de aceptación:** casillas que pasan o fallan, con valores, textos o comandos. Son la definición de terminado.
- **Fuera de alcance** (opcional, casi siempre útil): lo que el agente no debe "mejorar".

Para decidir si una línea entra, usa las dos preguntas de la documentación de buenas prácticas de Claude Code: ¿el agente lo podría encontrar leyendo el código? ¿De verdad iría a buscarlo? Si las dos respuestas son sí, déjala fuera.

PASO 01

La plantilla en blanco

Guárdala como `SPEC.md` en la rama, o pégala en un issue. Reemplaza cada texto entre `< >` y deja los encabezados como están: el script del paso 04 los busca por nombre. Las dos casillas fijas van en todas las especificaciones; la primera evita que el agente "arregle" un test fallido borrándolo.

```
# <Título: el cambio en pocas palabras>

## Objetivo
<Qué cambia, dónde y para quién. Una o dos frases, sin decir cómo construirlo.>

## Restricciones
- <Archivos o carpetas que puede tocar.>
- <Utilidades existentes que debe usar.>
- <Dependencias: ninguna nueva, o cuáles sí.>
- <Tu preferencia, si hay dos caminos válidos.>

## Criterios de aceptación
- [ ] <Entrada concreta: resultado exacto.>
- [ ] <Caso límite: vacío, error o máximo.>
- [ ] Ningún test existente se borra ni se salta.
- [ ] <Comandos que deben pasar, por ejemplo npm test y npm run typecheck.>

## Fuera de alcance
- <Lo que el agente no debe tocar ni "mejorar" en esta tarea.>
```

PASO 02

Ejemplo lleno: corregir un bug

Un fallo de una tienda de ejemplo, con montos ilustrativos: un cupón del 10% sobre $500 debería dejar $450, y aplicado dos veces deja $405. **Contexto del fallo** es una sección extra (puedes agregar las que necesites) y la primera restricción pide un test que falle antes del arreglo.

```
# Corregir: el cupón se descuenta dos veces

## Objetivo
Que un cupón aplicado al carrito se descuente una sola vez, aunque el cliente recargue la página.

## Contexto del fallo
- Pasos: agregar un producto de $500, aplicar el cupón DESC10, recargar la página.
- Esperado: total $450. Obtenido: total $405.

## Restricciones
- Primero un test que reproduzca el fallo; después el arreglo.
- Toca solo lib/cart/ y sus tests. No cambies cómo se guardan los cupones.

## Criterios de aceptación
- [ ] El test nuevo falla antes del arreglo y pasa después.
- [ ] Aplicar DESC10 y recargar dos veces deja el total en $450.
- [ ] Ningún test existente se borra ni se salta; npm test pasa.

## Fuera de alcance
- Cupones combinables y descuentos automáticos.
```

PASO 03

Ejemplo lleno: una función nueva

Una exportación de pedidos en el admin de una tienda de ejemplo; rutas y funciones ilustrativas. El orden de las columnas no está en el código, así que va escrito; RFC 4180 cubre los nombres con comas o comillas, y el 401, la ruta nueva que es fácil dejar abierta.

```
# Exportar pedidos a CSV desde el admin

## Objetivo
En /admin/pedidos, un botón descarga en CSV los pedidos del rango de fechas filtrado.

## Restricciones
- Reusa listOrders de lib/orders.ts; no escribas consultas nuevas.
- Columnas en este orden: número, fecha (ISO 8601), cliente, total, estado.
- La ruta exige la misma sesión de admin que la página. Sin dependencias nuevas.

## Criterios de aceptación
- [ ] Con 3 pedidos de prueba en el rango, el CSV tiene 1 encabezado y 3 filas.
- [ ] Un campo con coma o comillas sale entre comillas dobles, con las comillas internas duplicadas (RFC
```

```
4180).
- [ ] Sin sesión de admin, la ruta responde 401 y no devuelve datos.
- [ ] Ningún test existente se borra ni se salta; npm test y npm run typecheck pasan.

## Fuera de alcance
- Exportar a Excel o enviar el archivo por correo.
```

PASO 04

Revisar, construir y revisar otra vez

Antes de empezar, pasa `node check-spec.mjs SPEC.md`. El script completo está en el artículo (`ecomlabs.dev/g/spec`), sin dependencias y probado con Node 22: comprueba que las tres secciones tengan contenido y que ningún criterio diga "bien" o "correctamente". La plantilla y los dos ejemplos pasan.

Construye en una sesión nueva, con el contexto limpio. En Claude Code, `@` seguido de la ruta incluye el archivo:

```
Lee @SPEC.md. Antes de editar, dime qué archivos vas a tocar y cómo vas a comprobar cada criterio de aceptación. Al terminar, pega la salida de npm test y de npm run typecheck.
```

Al terminar, revisa con un revisor que no vio la conversación. No quites "omite el estilo": un revisor al que le pides problemas siempre encuentra alguno.

```
Usa un subagente para revisar el diff contra SPEC.md: comprueba que cada requisito esté implementado y que los casos límite listados tengan tests. Reporta los huecos que afecten la corrección y omite el estilo.
```

AL TERMINAR

Checklist

- [ ] El objetivo cabe en una o dos frases y describe el resultado.
- [ ] Las restricciones solo dicen lo que el agente no puede leer en el repo.
- [ ] Cada criterio pasa o falla, y uno prohíbe borrar o saltar tests.
- [ ] `check-spec.mjs` pasa antes de empezar.
- [ ] Pides la salida de los comandos y revisas el diff contra `SPEC.md`, empezando por los tests.
- [ ] Si cambias de opinión, actualizas primero la especificación.