EcomLabs

STEP-BY-STEP GUIDE

A spec for AI coding agents

The spec template with goal, constraints and acceptance criteria, two filled-in examples (a bug fix and a new feature), a Node script that checks the spec before you start, and the prompts to build and review. With a checklist.

FOR	Anyone working with coding agents such as Claude Code on web projects
TESTED WITH	Node 22.22.3 · check-spec.mjs on the template and both examples in this guide
UPDATED	November 19, 2026
DOCS	code.claude.com/docs/en/best-practices
ARTICLE	ecomlabs.dev/g/spec-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	The four sections
Step 01	The blank template
Step 02	Filled example: fixing a bug
Step 03	Filled example: a new feature
Step 04	Check, build, and check again
When you finish	Checklist

BEFORE YOU START

The four sections

- **Goal:** what changes, where and for whom. Describe the outcome, not the implementation.
- **Constraints:** what the agent cannot read in the repo: which files it touches, which utilities it uses, which dependencies stay out, and your preference when there are two valid approaches.
- **Acceptance criteria:** checkboxes that pass or fail, with values, copy or commands. They are the definition of done.
- **Out of scope** (optional, almost always useful): what the agent must not "improve".

To decide whether a line belongs, use the two questions from the Claude Code best practices docs: could the agent find this by reading the code? Would it actually go and look? If both answers are yes, leave it out.

STEP 01

The blank template

Save it as `SPEC.md` on the branch, or paste it into an issue. Replace every text between `< >` and keep the headings as they are: the script in step 04 looks for them by name. The two fixed checkboxes go in every spec; the first stops the agent from "fixing" a failing test by deleting it.

```
# <Title: the change in a few words>

## Goal
<What changes, where and for whom. One or two sentences, without saying how to build it.>

## Constraints
- <Files or folders it may touch.>
- <Existing utilities it must use.>
- <Dependencies: no new ones, or which ones are allowed.>
- <Your preference, if there are two valid approaches.>

## Acceptance criteria
- [ ] <Concrete input: exact result.>
- [ ] <Edge case: empty, error or maximum.>
- [ ] No existing test is deleted or skipped.
- [ ] <Commands that must pass, for example npm test and npm run typecheck.>

## Out of scope
- <What the agent must not touch or "improve" in this task.>
```

STEP 02

Filled example: fixing a bug

A bug in an example store, with illustrative amounts: a 10% coupon on $50 should leave $45, and applied twice it leaves $40.50. **Bug context** is an extra section (add whatever you need) and the first constraint asks for a test that fails before the fix.

```
# Fix: the coupon is discounted twice

## Goal
A coupon applied to the cart is discounted only once, even if the customer reloads the page.

## Bug context
- Steps: add a $50 product, apply the coupon SAVE10, reload the page.
- Expected: total $45. Actual: total $40.50.

## Constraints
- First a test that reproduces the bug; then the fix.
- Touch only lib/cart/ and its tests. Do not change how coupons are stored.

## Acceptance criteria
- [ ] The new test fails before the fix and passes after it.
- [ ] Applying SAVE10 and reloading twice leaves the total at $45.
- [ ] No existing test is deleted or skipped; npm test passes.

## Out of scope
- Stackable coupons and automatic discounts.
```

STEP 03

Filled example: a new feature

An order export in the admin of an example store; routes and functions are illustrative. The column order is not in the code, so it is written down; RFC 4180 covers names with commas or quotes, and the 401 covers a new route that is easy to leave open.

```
# Export orders to CSV from the admin

## Goal
On /admin/orders, a button downloads the orders in the filtered date range as a CSV file.

## Constraints
- Reuse listOrders from lib/orders.ts; do not write new queries.
- Columns in this order: number, date (ISO 8601), customer, total, status.
- The route requires the same admin session as the page. No new dependencies.

## Acceptance criteria
- [ ] With 3 test orders in the range, the CSV has 1 header row and 3 data rows.
- [ ] A field with a comma or quotes is wrapped in double quotes, with inner quotes doubled (RFC 4180).
```

```
- [ ] Without an admin session, the route returns 401 and no data.
- [ ] No existing test is deleted or skipped; npm test and npm run typecheck pass.

## Out of scope
- Exporting to Excel or emailing the file.
```

STEP 04

Check, build, and check again

Before you start, run `node check-spec.mjs SPEC.md`. The full script is in the article (`ecomlabs.dev/g/spec-en`), with no dependencies and tested with Node 22: it checks that all three sections have content and that no criterion says "properly" or "correctly". The template and both examples pass.

Build in a fresh session, with a clean context. In Claude Code, `@` followed by a path includes the file:

```
Read @SPEC.md. Before editing, tell me which files you will touch and how you will check each acceptance criterion. When you finish, paste the output of npm test and npm run typecheck.
```

When it is done, use a reviewer that never saw the conversation. Do not drop "skip style": a reviewer asked to find problems always finds something.

```
Use a subagent to review the diff against SPEC.md, check every requirement is implemented and the listed edge cases have tests, report gaps that affect correctness and skip style.
```

WHEN YOU FINISH

Checklist

- [ ] The goal fits in one or two sentences and describes the outcome.
- [ ] The constraints only say what the agent cannot read in the repo.
- [ ] Every criterion passes or fails, and one forbids deleting or skipping tests.
- [ ] `check-spec.mjs` passes before you start.
- [ ] You ask for the commands' output and review the diff against `SPEC.md`, tests first.
- [ ] If you change your mind, you update the spec first.