

Claude Code for Web Developers

7 practical workflows for building
and maintaining websites



Claude Code for Web Developers

Seven practical workflows for building and maintaining websites. A project-based workshop for developers who can already read HTML, CSS and JavaScript, and want a more reliable way to work with a coding assistant.

Build something you can inspect

The project is Fieldwork Studio: a fictional service website with a searchable catalog, an estimate calculator and a local-only request form. You will work from a deliberately incomplete starter to a tested reference implementation.

What is included

A runnable starter and reference project; 30 logic tests; a real boundary bug; a missing search feature; copyable task prompts; review gates; and a release worksheet. The companion creates all project files without downloading packages.

Edition 1.0 - 12 October 2026. Independent guide. Claude and Claude Code are Anthropic product names. This publication is not affiliated with or endorsed by Anthropic.

AI disclosure: the text, example code and cover were produced with AI assistance. Code and layout were checked using the procedures described in the validation record. The prompts are exercises, not transcripts of observed Claude sessions.

Read the contract first

The promise

By the end, you should be able to turn a small web task into a bounded request, judge the resulting diff and identify what remains unverified. You will practice these skills on a complete example instead of trusting a fluent completion message.

The prerequisites

You should know how to open a terminal, edit a file, read a JavaScript function and use a browser. Install Node.js 22 or later with npm, Git, and Claude Code using their official installation instructions. A supported Claude account or other supported access arrangement is required for the AI exercises; it is not included with this book. The demo and its tests work without Claude.

The limits

This is a front-end workshop, not a production commerce system. It has no authentication, database, email delivery, payment processing or real customer data. Its local development server is intentionally limited. The guide does not teach every Claude Code feature or certify a website as accessible or secure.

How to use the answers

Work in starter. Keep reference available for comparison, but try to state the behavior you need before opening the answer. Claude may choose different code. Judge it by behavior, readability and scope, rather than whether it matches the reference line for line.

There are two kinds of boxes: **TERMINAL** contains commands you run yourself; **PROMPT** contains a task you give Claude. Paths are relative to starter unless a different folder is named.

One project, seven decisions

The workflows build on the same starter folder. Finish the change and its checks before moving to the next one. The reference folder is a worked answer, not a second project that you must keep synchronized.

01 / Understand the repository / full guide p. 7

Find the entry points, run the baseline and define a project contract. Deliverable: a small, accurate CLAUDE.md and a task plan.

02 / Build the responsive page / full guide p. 10

Turn a visual brief into a constrained interface change. Deliverable: a usable layout at narrow and wide widths.

03 / Fix a boundary bug / full guide p. 13

Reproduce a discount failure before changing the code. Deliverable: correct pricing at and around the threshold.

04 / Add service search / full guide p. 16

Implement query and category filtering with an empty state. Deliverable: a feature with independent behavior checks.

05 / Protect the behavior / full guide p. 19

Add a meaningful invariant test and review a proposed refactor. Deliverable: a test that could catch a real regression.

06 / Check the interaction / full guide p. 22

Review labels, keyboard use, errors and reduced motion. Deliverable: an evidence log, with manual checks separated from automated ones.

07 / Prepare a release / full guide p. 25

Review the diff, package the right files and write a useful handoff. Deliverable: a release candidate and a reproducible verification record.

Reproduce before you repair

The pricing rule grants FIELD10 at a subtotal of \$1,000 or more. In starter, the implementation uses a strict greater-than comparison. A quote at exactly \$1,000 receives no discount. One character is wrong, but its effect appears in four test cases.

CODE / TERMINAL OR FILE EXCERPT

```
node --test tests/pricing.test.mjs
```

Expected starter result for this file: 15 tests, 11 passes, 4 failures. The failing cases include the exact threshold, a quantity that reaches it, rush at the threshold and coupon normalization at the threshold. A shared failure does not imply four separate bugs.

PROMPT / GIVE TO CLAUDE

Investigate the failing pricing tests. Do not edit yet. State the pricing contract from the code and instructions, reproduce the smallest failing input, and explain the root cause with a file reference. Distinguish the boundary bug from any other proposed improvement.

The smallest useful reproduction

CODE / TERMINAL OR FILE EXCERPT

```
calculateQuote({
  unitCents: 100000,
  coupon: 'FIELD10'
})
// Expected totalCents: 90000
// Starter actual totalCents: 100000
```

The input is intentionally synthetic: no catalog item costs exactly \$1,000. Unit tests can reach a boundary that the current interface does not expose. This protects the rule if catalog prices change later.

Do not fix the expected value to match the current implementation. The product contract is the reference for the assertion, and it explicitly includes the threshold.

Review the arithmetic, not just the total

PROMPT / GIVE TO CLAUDE

Fix the confirmed threshold bug with the smallest change. Keep the existing contract, rounding order and validation. Do not remove or weaken tests. Run pricing tests, then the full suite. Explain any remaining failures separately.

CODE / TERMINAL OR FILE EXCERPT

```
const eligible =
  coupon.trim().toUpperCase() === 'FIELD10' &&
  subtotalCents >= 100000;
const discountCents = eligible
  ? Math.round(subtotalCents * 0.10) : 0;
const discountedCents = subtotalCents - discountCents;
const rushCents = rush
  ? Math.round(discountedCents * 0.20) : 0;
const totalCents = discountedCents + rushCents;
```

For 100000 cents with the coupon and rush: discount is 10000, the discounted base is 90000, rush is 18000 and the final total is 108000. Applying rush to the original subtotal would incorrectly produce 110000 after subtracting the discount.

At 100005 cents, round the discount to 10001. The remaining 90004 gives a rush adjustment of 18001. The final total is 108005. The reference rounds each adjustment separately; it does not round only once at the end.

Why integer cents matter here

Keep money in cents while calculating. Convert to a formatted string only when displaying it. This avoids parsing formatted currency back into arithmetic. It is a teaching model for one currency with two decimal places, not a universal accounting engine.

Expected after this fix: pricing 15/15; full suite 24/30. The six search failures still belong to workflow 04.

Check the boundary on both sides

A passing example at the threshold is necessary but insufficient. You also need the adjacent cases and the interaction with other rules. Otherwise a patch that always grants the discount could make the original failure disappear while introducing a different error.

The review matrix

99999 cents with FIELD10: no discount. 100000 with FIELD10: 90000 total.
100001 with FIELD10: 90001 total. 100000 without the coupon: unchanged. 50000
times two with FIELD10: 90000. Add rush to that final case: 108000.

CODE / TERMINAL OR FILE EXCERPT

```
import assert from 'node:assert/strict';
import { calculateQuote } from '../src/pricing.mjs';
const q = calculateQuote({
  unitCents: 100001, coupon: 'FIELD10'
});
assert.equal(q.totalCents, 90001);
```

Inspect the diff

The required behavior fix is the comparison operator. Unrelated formatting across the entire project makes that harder to review. Ask Claude to separate optional cleanup from the bug fix. A tiny diff with clear evidence is often a stronger result than a large refactor.

If Claude cannot reproduce it

Check the working folder and confirm it is using starter rather than reference. Compare the exact command and input. If a test runner reports zero tests, inspect the file path before accepting a green exit. If the behavior changes only in the browser, restart or refresh the correct local server.

Checkpoint: record the reproduction, the root cause, the changed line and the before/after test results. This four-part note is enough for another developer to understand the fix.

What the full guide includes

The full guide follows one fictional service website through seven workflows: repository mapping, responsive layout, a pricing repair, service search, regression tests, interaction review and release preparation.

A runnable project, not just prompt lists

The companion creates both starter and reference folders. It includes 30 logic tests, the known pricing and filtering defects, a local development server and a complete answer to compare. No external package installation is needed.

Designed for working developers

You should already be able to read HTML, CSS and JavaScript. Claude access is separate from the purchase. The project is a front-end learning demo and does not process payments or send email.

This sample contains introductory material and the complete pricing-debugging workflow. The full book includes the remaining workflows, task cards, troubleshooting notes, source links and the edition validation record.

AI disclosure: text, example code and cover were produced with AI assistance. The book distinguishes executed code checks from prompts that readers will run themselves. Independent guide; not affiliated with or endorsed by Anthropic.