EcomLabs

STEP-BY-STEP GUIDE

How to report a bug to AI

A template for asking an AI to fix a bug without making it guess: the exact error, how to reproduce it, where to look and the cause before the change. With three filled-in examples: frontend, API and a Shopify store.

FOR	Claude Code, Cursor or any agent that reads your code
TESTED WITH	Node 22.22 · cart, React and API examples run locally
UPDATED	October 30, 2026
DOCS	platform.claude.com · code.claude.com/docs
ARTICLE	ecomlabs.dev/g/bug-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Step 01	Copy the template
Step 02	Fill in the exact error
Step 03	Write the reproduction
Step 04	Say where to look
Step 05	Ask for the cause, then the fix
Example 1	Frontend: a React list
Example 2	API: JSON that is not JSON
Example 3	Shopify store: the wrong size goes in
When you are done	Checklist

STEP 01

Copy the template

It gives the AI what it cannot guess and asks for the cause before any code changes. Fill it in order; if a section does not apply, say why ("there is no error message"): that is information too.

```
## Error
<full message, not a summary>
<stack trace or full output>

## How to reproduce
Command or steps: <npm test, a URL, the clicks in order>
What I did: <...>
What I expected: <the right result, with numbers if any>
What happened: <what you saw>
Always or sometimes?: <always / sometimes / only in production>

## Where to look
File or function: <...>
Recent change: <commit, PR or what was added>
Already ruled out: <...>

## How I want you to work
1. Before you change any code, explain the cause. Do not edit yet.
2. Once I confirm, fix the cause, not the symptom.
3. Add a test that fails today and passes with the fix.
4. Run <test command> and paste the output.
```

STEP 02

Fill in the exact error

Copy the text, do not summarise it. The exact message can be searched for in the code; "something about undefined" cannot. Where to find it:

WHERE IT HAPPENS	WHAT TO COPY
Browser	The DevTools console text, with its "at …" lines
A request	From the Network tab: URL, status code, `content-type` and response body
Terminal or tests	The command's full output; with Claude Code you can pipe it in: `cat error.log | claude -p "..."`
Server	The log lines around the error, with their timestamps

In the stack trace, the most useful line is the first one that names a file in your project.

> **CAUTION**
> Before you paste logs or responses, strip tokens, passwords, API keys and customer data. An error does not need a secret's real value to make sense: replace it with `<TOKEN>`.

STEP 03

Write the reproduction

- **What you did**: the exact steps, in order. If there is data, the data (the SKU, the postcode, the test user).
- **What you expected**: the right result. It tells the AI when it is done.
- **What happened**: what you saw, before interpreting it.
- **Always or sometimes?** An intermittent error usually points at timing, caching or data rather than at one line.

If you can, turn it into a failing test. That is a reproduction the AI can run on its own, before and after the fix.

STEP 04

Say where to look

What you know and the code does not say: what changed, since when it fails, which part is new. In Claude Code, type `@` and a path to include the file. If you use git, `git log --oneline -5` lists the last commits so you can name the suspect. If you have no idea, say so and ask it to investigate without editing.

STEP 05

Ask for the cause, then the fix

When the AI explains the cause, check it: does it account for the error and the result you saw? If it does, ask for the fix:

```
The cause makes sense. Fix the cause, not the symptom.
Add a test that fails today and run npm test when you are done.
Paste the command you ran and its output.
```

If the explanation does not match what you saw, say so with the detail that does not fit, before any new code exists.

EXAMPLE 1

Frontend: a React list

The component loads products in a `useEffect`. We checked the error message locally by rendering the component with `react-dom/server`; the "at" line and the steps are illustrative.

```
const [products, setProducts] = useState();
// ...
return <ul>{products.map((p) => <li key={p.id}>{p.name}</li>)}</ul>;
```

```
## Error
TypeError: Cannot read properties of undefined (reading 'map')
    at ProductList (ProductList.jsx:10:23)

## How to reproduce
Steps: open /products.
Expected: the product list.
Happened: blank screen and the error in the console. Always, on first load.

## Where to look
src/components/ProductList.jsx. It started when we moved loading into useEffect.
```

The cause: on the first render, `useEffect` has not run yet and `products` is `undefined`. The fix belongs in the initial state: we checked it, and with `useState([])` the component renders without an error. If you also want a "Loading…" state, ask for it separately: that is another decision, not part of the bug.

EXAMPLE 2

API: JSON that is not JSON

We ran this one locally on Node 22: a server that only has `/api/shipping` and a client that calls `/api/ship`.

```
const res = await fetch(`/api/ship?zip=${zip}`);
const data = await res.json();
```

```
## Error
SyntaxError: Unexpected token '<', "<!DOCTYPE "... is not valid JSON
In Network: GET /api/ship?zip=10001, status 404, content-type text/html

## How to reproduce
Steps: at checkout I enter the zip code 10001.
Expected: the shipping cost ({"cost": 990}).
Happened: no cost shows up and the error is in the console. Always.

## Where to look
The call is in src/checkout/shipping.js. It started after we reorganised
the API routes.
```

The key detail is the Network line: the `JSON.parse` error says the response starts with `<`, so it is HTML. The 404 says why: the URL does not exist. Without that line, the AI might start "fixing" the parsing. In our run, the correct URL answered `200 { cost: 990 }`. It is worth asking for the client to check `res.ok` before calling `res.json()`, too.

EXAMPLE 3

Shopify store: the wrong size goes in

An illustrative example: we did not run it on a real store. The theme's product form keeps the variant in a hidden input; we validated this Liquid line with Theme Check:

```
<input type="hidden" name="id" value="{{ product.selected_or_first_available_variant.id }}">
```

```
## Error
No message. In Network, the POST to /cart/add.js sends id 44100000000001,
which is size S. Size M is 44100000000002 (illustrative ids).

## How to reproduce
Steps: open /products/basic-tee, pick size M, click "Add to cart".
Expected: "Basic tee - M" in the cart.
Happened: size S goes in. Always, with any size but the first.

## Where to look
The hidden name="id" input in the product form and the variant picker's
JavaScript. It started when we swapped the dropdown for buttons.
```

There is no error to copy here, so the evidence is the request: which id was sent and which one should have been. The input starts with the first available variant; when the shopper picks another size, the theme's JavaScript has to update that value. If the new picker does not, the form keeps sending the starting id.

> **CAUTION**
> Test the fix on an unpublished theme or a development store, never on the live theme.

WHEN YOU ARE DONE

Checklist

- [ ] The error message is complete, copied as text.
- [ ] The stack trace is there, or the request with its status and response.
- [ ] You wrote what you did, what you expected and what happened.
- [ ] You said whether it happens always or sometimes.
- [ ] You named the file, function or recent change.
- [ ] You stripped tokens, passwords and customer data.
- [ ] You asked for the cause before the change.
- [ ] You asked for a test that fails today and for the test output.