

12 · Code and platforms

Bug Diagnosis and Fix Request Template

Report a bug and request a fix from a developer or AI tool: reproduction steps, expected vs actual, safe logs, environment, constraints, regression check.

For: Project owners and teams reporting bugs in their sites or tools, and anyone asking an AI tool for a fix who wants a precise repair that does not break something else.

Level: Beginner Version 1.0 · 2026-09-21

"The site doesn't work" is not an error report; it is the start of a long investigation. A developer or AI tool given a vague description guesses, and guessing produces changes in places unrelated to the problem, sometimes breaking what used to work. A good report answers in advance every question the fixer will ask: how do I see the bug myself? What should have happened? Where, and on which device? And what must not be touched during the fix?

Rule one: no secrets in prompts or reports. Never paste API keys, passwords, access tokens, database connection strings, session cookies or real customer data into a chat with an AI tool, a bug report, or a message to a developer. Replace them with fake values of the same shape. If you paste a real key by mistake, treat it as exposed: rotate it with the provider immediately; deleting the message is not enough.

The nine parts of a report

1. Summary

One line saying what happens and where: "The Confirm order button does nothing on iPhone on the checkout page." Not "problem with the site".

2. Reproduction steps

Numbered steps that someone who does not know the project can follow and see the bug for themselves. Start from a clear point (page URL, signed-in state) and list the data you entered (fake or public). Run the steps again yourself before sending: does it happen always, sometimes, or once?

3. Expected vs actual

Two separate sentences: what should have happened and what actually happened. Many "bugs" turn out to be a difference in expectations; writing the expectation reveals that early.

4. Environment

Device, operating system, browser and version, app or site (test or live), account or role (no password), the approximate time of the bug and its time zone, and whether it started after a particular change.

5. Evidence and safe logs

A screenshot or short recording, the error message text exactly as shown (copied, not retyped), and the relevant lines from the log or the browser's developer console. Before sending, clean the logs:

- ☐ Keys and tokens are replaced with [REDACTED] or a fake value of the same shape.
- ☐ Customer names, numbers, addresses and emails are hidden or replaced.
- ☐ Database connection strings, session cookies and authorisation headers are removed.
- ☐ Screenshots show no saved passwords, private notifications or customer data.
- ☐ You sent only the lines related to the bug, not the whole log file.

6. Minimal example

The smallest version of the problem you can show: only the file or function involved, with the least code and data that still shows the bug. Building the minimal example sometimes reveals the cause before you even send the report. To get there, remove unrelated parts step by step, checking after each removal that the bug still appears.

7. What you tried

What you have done so far and the result: "Cleared browser cache: no change", "Works on Chrome", "Started after Tuesday's update". This saves the fixer from repeating what was already tried.

8. Fix constraints

What must not be touched or changed: "Do not change payment logic", "Do not add new libraries", "Keep right-to-left direction", "Do not change the database structure". AI tools tend to rewrite more than needed; constraints keep the fix where it belongs.

9. Regression check

After the fix it is not enough for the bug to disappear; no new bug should appear. Write in advance: the same reproduction steps must now give the expected result, a short list of what must keep working (features close to the fix), and, where possible, an automated test that stops the bug from returning.

Bug report template

Bug report

Summary: [what happens and where, in one line]

Severity: [blocks work / partial / cosmetic] Frequency: [always / sometimes / once]

Steps to reproduce:

1. [Open URL .. with an account in role ..]
2. [..]
3. [..]

Expected: [..]

Actual: [..]

Environment: device [..] | OS [..] | browser and version [..]

environment [test / live] | role [..] | time and time zone [..]

started after: [change or update, if any]

Evidence (cleaned): [screenshot / recording / error text / log lines]

Minimal example: [file or function + least data that shows the bug]

What I tried: [action → result]

Fix constraints: [what must not change]

Regression check: [what must keep working after the fix]

Fix request prompt for an AI tool

Fix request prompt

You are a careful developer. Below are a bug report and a minimal code example.

In order:

- 1) Explain the most likely cause in 3 lines;
say if you are unsure and what would confirm it.
 - 2) Propose the smallest change that fixes the bug, in the listed files only.
 - 3) Follow the fix constraints literally.
If a constraint blocks the fix, say so; do not bypass it.
 - 4) Write a test that fails before the fix and passes after it.
 - 5) List what the change might affect so I can test it by hand.
- Do not rewrite whole files, add libraries, or change formatting or names outside the fix.

Report: [paste the report]

Code: [paste the minimal example after replacing any key or real data
with fake values]

After the fix

1. **Read the change.** Is it small and in the expected place? Did it respect the constraints? If it touched files that were not mentioned, ask why before accepting.
2. **Repeat the reproduction steps** on the same environment and device where the bug appeared, not just on your own machine.
3. **Run the regression check:** nearby features and all automated tests.
4. **Close the report with two lines:** the actual cause and what changed. This record shortens diagnosis if something similar happens again.

Completed example

Illustrative example

Lamsa Store is a fictional online shop in Tripoli selling handmade care products. The owner received messages from two customers saying "the order won't send".

The report as sent

Summary: "Confirm order" does nothing on iPhone at checkout when a delivery slot is chosen.

Severity: blocks work (order cannot be completed) Frequency: always on iPhone

Steps to reproduce:

1. Open the shop in Safari on iPhone, not signed in.
2. Add "Olive oil soap" to the basket and tap "Checkout".
3. Fill name and phone with test values, choose delivery slot "Tomorrow 10:00".
4. Tap "Confirm order".

Expected: the "We received your order" page appears with an order number.

Actual: nothing happens. The button does not change and no message appears.

Environment: iPhone | up-to-date iOS | Safari | live site | guest, no account
started after the "Delivery slot" field was added last week.

Evidence: 15-second screen recording. From the developer console (cleaned):

RangeError: Invalid time value at checkout.js:88

POST /api/orders not sent

Authorization: Bearer [REDACTED]

What I tried: works on desktop Chrome and on Android. Works on iPhone with no slot.

Fix constraints: do not change payment logic or total calculation. No new libraries.

Keep right-to-left direction in the form.

Regression check: order without a slot, order with a slot on Chrome and Android,
the error message when phone is left empty.

The minimal example attached (the relevant function only):

Minimal example

```
// checkout.js (excerpt)
function buildOrder(form) {
  const slot = form.deliveryDate + ' ' + form.deliveryTime; // "2026-09-22 10:00"
  return {
    items: form.items,
    deliverAt: new Date(slot).toISOString(),           // line 88
  };
}
```

The tool's reply, summarised: the most likely cause is that a date string with a space between day and time is not understood by some Safari versions, producing an "invalid date" and stopping the code before the order is sent, while other browsers accept it. Proposed fix: build the date in the

standard format with a T between day and time, check the date is valid, and show the customer a clear message instead of failing silently. Two lines changed inside the same function, plus a test that passes the old format and checks the result is a valid date.

Verification: the owner repeated the steps on an iPhone: the confirmation page appeared. She then ran the three regression checks, all fine. The report was closed with two lines: "Cause: non-standard date format rejected by Safari. Change: standard format + visible error message + test."

Common mistakes

- **Mistake:** pasting the environment file or a full log into the chat "for context". **Fix:** send only the related lines after cleaning them, and replace every secret with a fake value. If a leak happens, rotate the key immediately.
- **Mistake:** describing the fix you imagine instead of the problem ("change the button"). **Fix:** describe what happens and what was expected; leave the diagnosis to the fixer.
- **Mistake:** sending the whole project. **Fix:** a minimal example; a tool that reads everything edits everything.
- **Mistake:** retyping the error message from memory. **Fix:** copy it exactly; one different word can change the diagnosis.
- **Mistake:** accepting the fix because the bug is gone on your machine. **Fix:** verify on the device and environment where it appeared, and run the regression check.
- **Mistake:** asking for several bugs to be fixed in one message. **Fix:** one report per bug, one fix per report.

Completion checklist before sending

- ☐ The summary is one line stating what and where.
- ☐ Reproduction steps were run twice, and you know whether the bug is constant or intermittent.
- ☐ Expected and actual are written separately.
- ☐ The environment is complete: device, browser, version, environment, role and time.
- ☐ Evidence is attached, and the error message is copied exactly.
- ☐ No keys, passwords, tokens or customer data anywhere in the report or prompt.
- ☐ The minimal example actually shows the bug.
- ☐ Fix constraints are written.
- ☐ The regression check is defined in advance.
- ☐ One report for one bug.

About the author — Mohamad Charafeddine is an entrepreneur from Tripoli, Lebanon: founder of the Passion digital marketing agency, president of Takaful Youth Association, and founder of digital platforms. He trains individuals and teams in applied AI and marketing.

Examples in this file are fictional. Free to use in your work and organisation; credit the source if you republish.
mohamadcharafeddine.com/en/resources/bug-report-fix-request/