

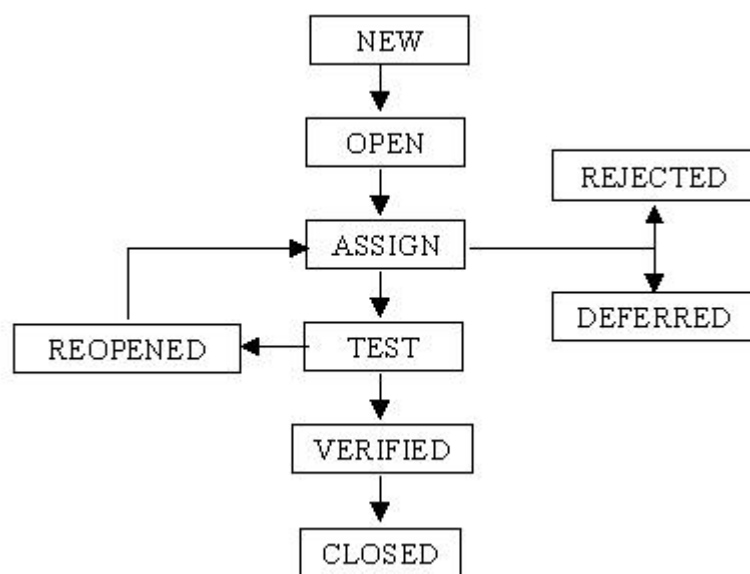
## Introduction

A defect is a specific concern about the quality of an Application Under Test (AUT). Defect can be defined as an inconsistency in the behavior of the software. No software exists without a defect or bug. The elimination of bugs from the software depends upon the efficiency of testing done on the software. A defect is basically the difference between the expected result and the actual result.

The cost of finding and correcting defects represents one of the most expensive software development activities. For the foreseeable future, it will not be possible to eliminate defects. While defects may be inevitable, we can minimize their number and impact on our projects. To do this, development teams need to implement a defect management process that focuses on preventing defects, catching defects as early in the process as possible, and minimizing the impact of defects. A little investment in this process can yield significant returns.

## Defect Life Cycle

The bug has a life cycle in software development process. The bug should go through the life cycle to be closed. A specific life cycle ensures that the process is standardized. The bug attains different states in the life cycle. The life cycle of the bug can be shown diagrammatically as follows:



The different states of a bug can be summarized as follows:

1. New
2. Open
3. Assign
4. Test
5. Verified
6. Deferred
7. Reopened
8. Duplicate
9. Rejected
10. Closed

## Description of Various Stages

1. **New** : When the bug is posted for the first time, its state will be "NEW". This means that the bug is not yet approved.
2. **Open** : After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state to "OPEN".
3. **Assign** : Once the lead changes the state to "OPEN", he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to "ASSIGN".
4. **Test** : Once the developer fixes the bug, he has to assign the bug to the testing team for the next round of testing. Before he releases the software with the bug fixed, he changes the state of the bug to "TEST". It specifies that the bug has been fixed and is released to the testing team.
5. **Deferred** : When the bug is changed to the deferred state, the bug is expected to be fixed in future releases. The reasons for changing the bug to this state has many factors. Some of them are the priority of the bug may be low, a lack of time for the release or the bug may not have a major effect on the software.
6. **Rejected** : If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "REJECTED".
7. **Duplicate** : If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to "DUPLICATE".
8. **Verified** : Once the bug is fixed and the status is changed to "TEST", the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "VERIFIED".
9. **Reopened** : If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "REOPENED". The bug traverses the life cycle once again.
10. **Closed** : Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "CLOSED". This state means that the bug is fixed, tested and approved.

Defect prevention is much more effective and efficient in reducing the number of defects; most organizations conduct defect discovery and removal. Discovering and removing defects is an expensive and inefficient process. It is much more efficient for an organization to conduct activities that prevent defects.

## Guidelines on deciding the Severity of Bug

It indicates the impact each defect has on testing efforts or users and administrators of the application under test. This information is used by developers and management as the basis for assigning priority of work on defects. "Severity" is associated with standards. "Severity" is the state or quality of being severe; severe implies adherence to rigorous standards or high

principles and often suggests harshness; severe is marked by or requires strict adherence to rigorous standards or high principles, e.g. a severe code of behavior.

### **Guidelines for assignment of Priority Level**

"Priority" is associated with scheduling. "Priority" means something is afforded or deserves prior attention; precedence established by order of importance (or urgency).

A sample guideline for assignment of Priority Levels during the product test phase includes:

1. **Critical / Show Stopper** : An item that prevents further testing of the product or function under test can be classified as a Critical Bug. No workaround is possible for such bugs. Examples of this include a missing menu option or security permission required to access a function under test.
2. **Major / High** : A defect that does not function as expected/designed or that causes other functionality to fail to meet requirements can be classified as a Major Bug. The workaround can be provided for such bugs. Examples of this include inaccurate calculations; the wrong field being updated, etc.
3. **Average / Medium** : The defects which do not conform to standards and conventions can be classified as Medium Bugs. Easy workarounds exist to achieve functionality objectives. Examples include matching visual and text links which lead to different end points.
4. **Minor / Low** : Cosmetic defects which does not affect the functionality of the system can be classified as Minor Bugs.

**Defect Management Process** The defect management process is based on the following general principles:

- The primary goal is to prevent defects. Where this is not possible or practical, the goals are to both find the defect as quickly as possible and minimize the impact of the defect.
- The defect management process should be risk driven -- i.e., strategies, priorities, and resources should be based on the extent to which risk can be reduced.
- Defect measurement should be integrated into the software development process and be used by the project team to improve the process. In other words, the project staff, by doing their job, should capture information on defects at the source. It should not be done after-the-fact by people unrelated to the project or system.
- As much as possible, the capture and analysis of the information should be automated.
- Defect information should be used to improve the process. This, in fact, is the primary reason for gathering defect information.
- Most defects are caused by imperfect or flawed processes. Thus to prevent defects, the process must be altered.

## How to write a good bug report

Here is the process on how to write a bug report:

1. **Issue/Defect/Bug ID**
2. **Heading/ Title**
3. **Summary**
4. **Screenshots/ Images/ Video recordings**
5. **Expected results vs. actual results**
6. **Step-by-step procedure to find the bug**
7. **Environment**
8. **Console logs**
9. **URL of the Source**
10. **Priority and severity**
11. **Additional info**

**Issue ID:** Maintain a clear issue ID. You can auto-generate issue IDs with a bug tracking tool. This will also help you avoid duplication.

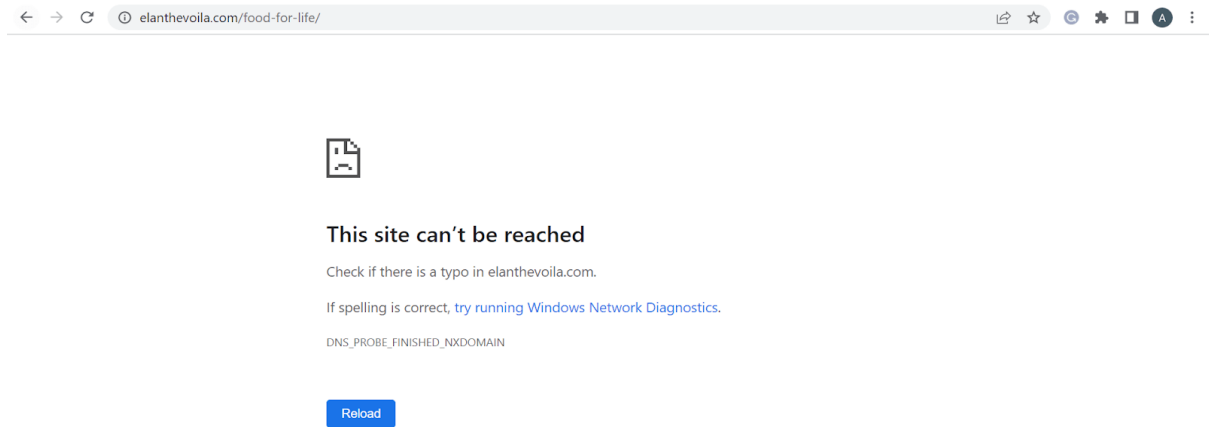
**Heading:** Your heading catches the attention of your developer first. Make it short, clear, and crisp with the needed info on category, pages, or location. Take a look at this section on how to write a good bug report example:

*Example: “Homepage: The new blog link doesn’t work”*

**Summary:** Elaborate on your bug report heading with the necessary points on how and when the bug has been found. A good report summary can come in handy when your developer wants to locate the bug in the bug inventory in the future.

*Example: “We have published the social media posts at 10.15 AM on our recent blog titled “Food for life”. But when I clicked on the blog link, nothing happened. I tried to visit the page manually, but it shows ‘The site can’t be reached”*

**Screenshots/Images/Video recordings:** It’s always better to go visual since visualization can yield better results. Yes, [visual testing](#) is of great importance. A video recording, image, or screenshot can soon help your developer locate the bug. They would surely be able to know how exactly the bug has impacted and what they need to do next.



**Expected results vs Actual results:** Now, convey what you expected would appear on the screen and what appeared to your developer. This will help any developer to get a clear picture of what's expected and what's not. This is where a test case comes into play. The actual difference between a test case and a bug report is that the test case differentiates between the expected result and the actual result. On the other hand, a bug report speaks about the number of errors and how to fix them.

*Example:*

*Expected result: The blog link would take the user to the webpage.*

*Actual result: The blog link doesn't take the user to the webpage.*

**Step-by-step procedure to find the bug:** Assume that your developer is a noob who doesn't know how and when you found the bug. How would you explain it to them? First, illustrate it in the report.

*Example:*

1. *Go to the homepage.*
2. *Click on the resources page.*
3. *Click on the blog.*
4. *Click on this blog link: <https://www.elanthevoila.com/blog/food-for-life/>*

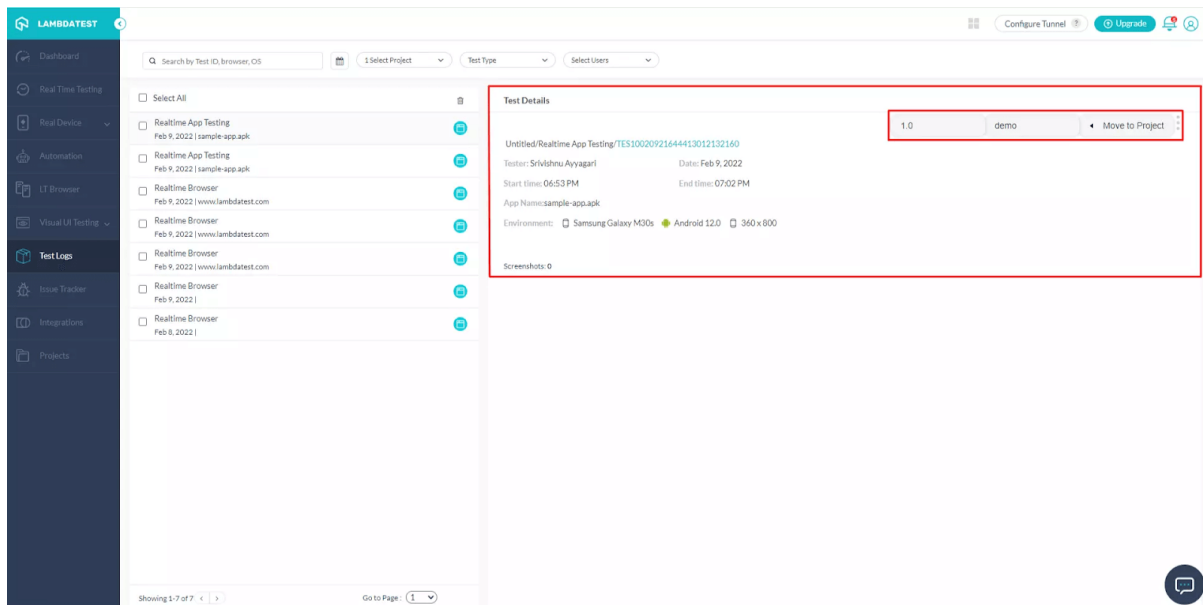
**Environment:** Include the vital info such as browser, OS version, size of the screen, pixel ratio, and level of zooming in the [test environment](#). This will allow the developer to get an idea on which platform or gadget the error appears to be zoomed.

*Example:*

- *Browser: Firefox 100*
- *Screen size: 1920×1080*

- *OS: Windows Server 2022*
- *Viewport Size: 360 x 800*
- *Zoom level: 100%*
- *Pixel ratio: @3x*

**Console logs:** Console logs are the area where a developer can witness what errors have been identified on the webpage from a technical point of view. This is the best choice to identify where it all went wrong after you get an idea on how to write a bug report.



**URL of the Source:** Your developer needs to know the exact location of where the bug was found. This will help them take action within minutes.

*Example: “<https://www.elanthevoila.com/blog/food-for-life/>”*

**Priority and severity:** Give a clear picture of how severe the bug is. Prioritize it accordingly. The [bug severity](#) can range anywhere between

- Major
- Critical
- Trivial
- Enhancement
- Minor

The severity can range anywhere between high, medium, or low.

**Additional info:** It’s always good to align the document professionally. Ensure that you provide a few extra pieces of info too. For example, give information on your name, due date, assigned developer, and conversation with the user or customer needed to [fast-track the debugging process](#).

*Example:*

*Name: Swapnil Biswas*

*Due date: 02/08/2023*

*Assigned developer: Brandon Stark*

**How do you write a good bug report? What information should you include, and what's the best way to report a bug? Fear not: in this blog post, we'll explore the best practices for bug reporting.**

So, you've just told your developer team that you found a bug. Instantly, you hear the dreaded reply... "I need more information."

Before you know it, you've got an email thread a million miles long.

You can avoid this situation by writing better, more detailed bug reports from the get-go.

Not to worry: we'll share some tips and tricks on how to write actionable bug reports that will make your developers love you!

## **What is a bug report?**

Let's start with the basics: what is an effective bug report?

Bug reports are an unavoidable part of the software development life cycle. They provide information about a problem or issue that occurred in a website or application.

An effective bug report will include a title, bug description, environment information, steps to reproduce, and more—so developers can reproduce and fix the problem.

A high-quality bug report makes all the difference in how fast the bug gets resolved—so let's have a look at how to do it right!

## **How to write a good bug report**

Developers are often under a ton of pressure to solve issues quickly without actually having a lot of time on their hands.

They usually face two extremes: *too much unhelpful information* or *too little important information*.

For every bug report, we highly recommend using a [bug tracker](#) like Trello, Jira, Asana, GitHub or GitLab and implementing a consistent, standardized approach.

Here's how to write a bug report:



## 1. Title/Bug ID

Keep it short and specific.

Make sure it clearly summarizes what the bug is. Having a clear title on your bug report makes it easier for the developer to find later on and merge any duplicates.

Examples:

✗ Bad: *"I can't see the product when I add it, for some reason I try and it doesn't. WHY? Fix it asap."*

- vague
- aggressive
- too verbose
- asks for a solution to be implemented

✓ Good: *"CART - New items added to cart do not appear"*

- it helps developers instantly locate the issue (CART)
- it focuses on the actual technical problem

When developers review it, they'll be able to instantly assess the issue and move on to other elements of the bug report.

## 2. Summary

If your title isn't enough, you can add a short report summary. And we mean short.

In as few words as possible, include when and how the bug occurred.

Your title and description may also be used in searches, so make sure you include important keywords.

Examples:

✗ Bad: *"The other day I was trying to add stuff to test and nothing showed up when I did that or clicked on the button."*

✓ Good: *"On [DATE], I tried adding [PRODUCT] to the cart, nothing happened after clicking the 'add' button on the product overview webpage."*

✗ Bad: *"The design text on the pricing page looks super weird and doesn't seem right. It shouldn't look that big and should be in a different color."*

✓ *Good: The headline text size and color on the pricing page don't match the original designs.*

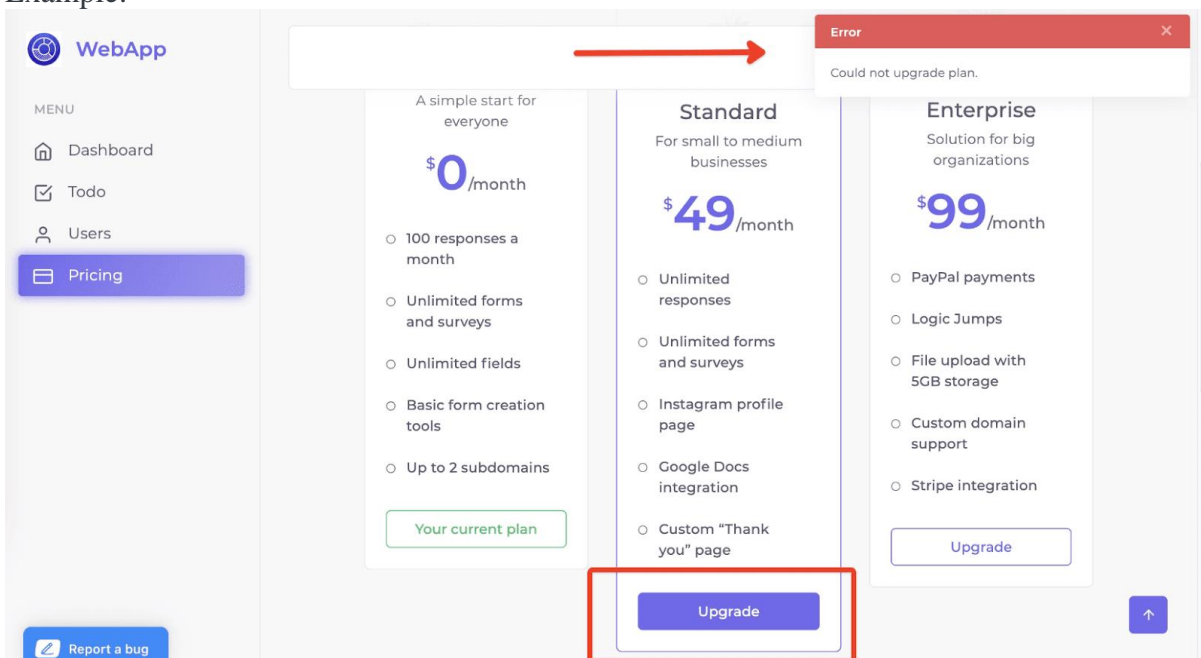
### 3. Visual proof/screenshot

We all know that a picture is worth a thousand words. That stands true for bug reporting.

While it may not tell the whole story, a screenshot or video can add a lot of value by getting your developers to see and understand the problem faster.

[Website annotation tools](#) go a long way here to help you drive your point across.

Example:



### 4. Expected vs. actual results

When you report a bug, take some time to explain to your developer what you expected to happen and what actually happened.

Example:

Expected result: *"Item should be added to the cart when I click ADD"*

Actual result: *"Item does not appear in the cart"*

## 5. Steps to reproduce

Here is your opportunity to share the steps needed to recreate the bug!

Always assume that your developer has no idea about the bug you found—how does he reproduce it?

As always, keep it simple!

The steps to follow should be comprehensive, easy to understand, and short.

The most important goal of this step is for your developer to experience the bug first-hand.

Use a numbered list here. And if you've already managed to recreate the issue several times, you can include the reproducibility rate (example: 12/12 times bug reproduced).

Example:

1. *Search for product XYZ.*
2. *Click on product XYZ in search results.*
3. *Click on "Add to Cart" button.*
4. *Go to cart.*

Want to go one step further? Add a short video recording, or use a session replay tool—so the developer can watch you reproduce this new bug!

## 6. Environment

Websites and apps can behave very differently depending on the environment used.

It's critical that the following information be included in any report you share:

- Browser (Chrome, Firefox, Safari...)
- Operating system (OS) and version (Mac, Windows...)
- Screen size
- Zoom level
- Pixel ratio

<b>Browser</b>	Chrome 88.0.4324.192
<b>Screen Size</b>	1280 x 800
<b>OS</b>	OS X 11.2.1
<b>Viewport Size</b>	1223 x 664
<b>Zoom Level</b>	100%
<b>Pixel Ratio</b>	@2x

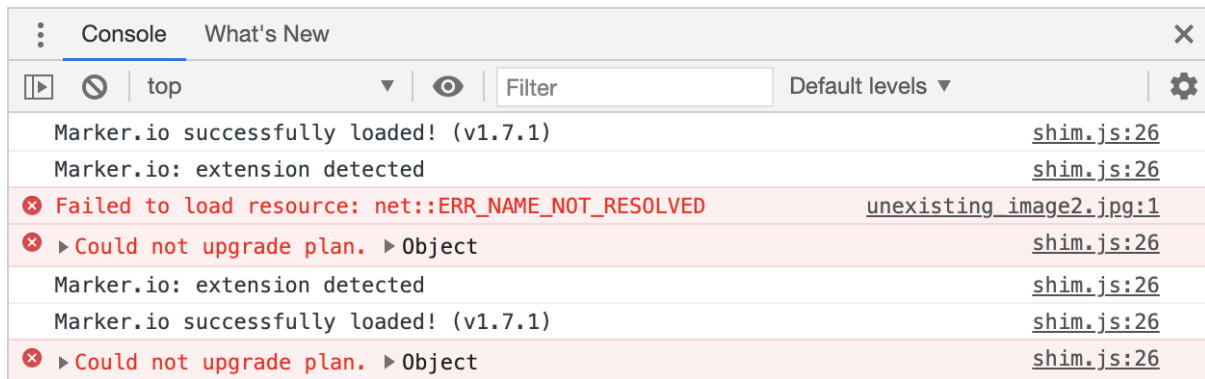
Pro tips:

- Marker.io will automatically collect this technical information
- Use [BrowserStack with Marker.io](#) to automatically grab virtual device information and capture cross-browser issues
- Additional info that can be helpful: device type, network connectivity, and battery state

## 7. Console logs

These logs show developers all errors that occur on a given webpage.

Logs can also include info that track certain user actions.



In general, including console logs can be valuable for developers as it can help them dig deeper and identify the root of the problem.

When debugging, it saves them a bunch of time on any issue!

A lot of crashes or errors are hard to replicate, so having the logs can be super informative.

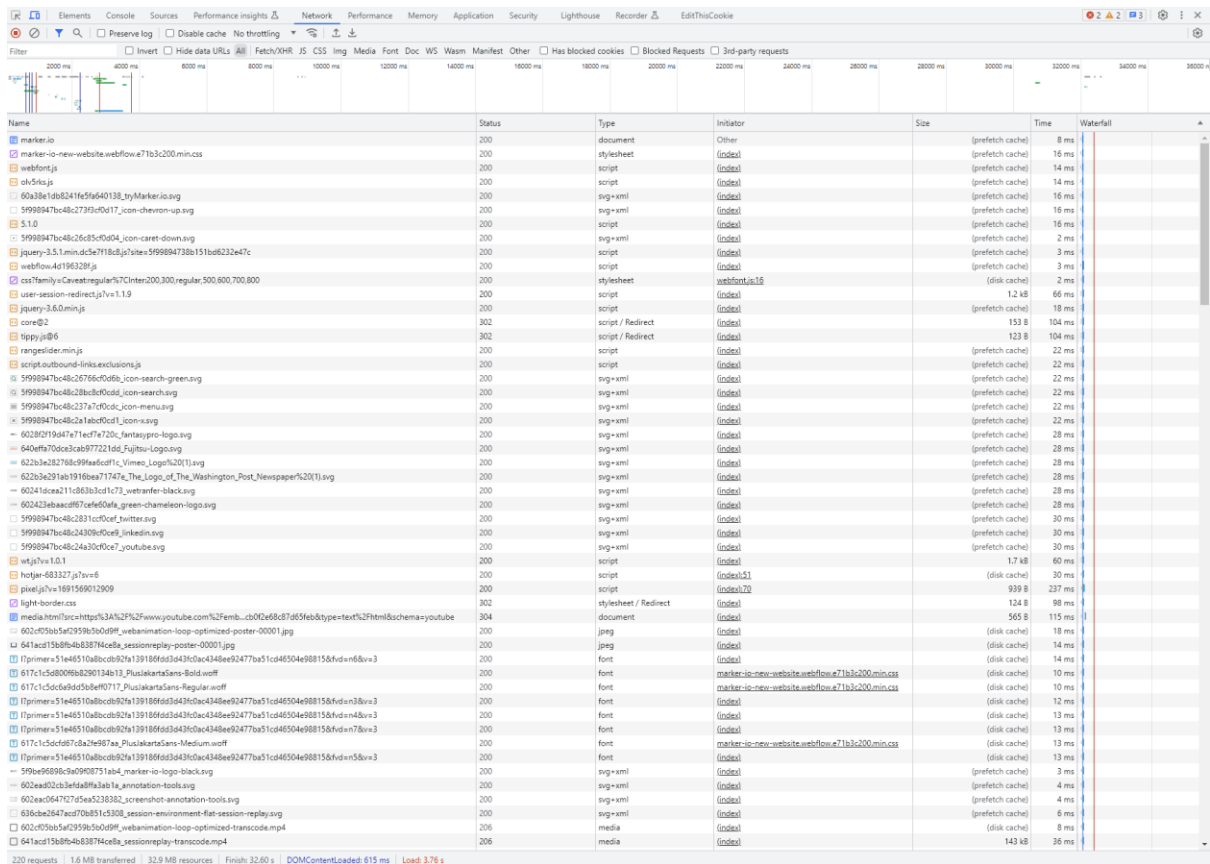
Pro tip: Marker.io adds console logs automatically to every ticket you create.

## 8. Network requests

Network requests detail the flow of data between client and server.

Inspecting network requests will help developers investigate failed API calls, retrieval of resources (like images), and delayed responses—all things that could slow down a webpage, for example.

It's invaluable when trying to fix bugs like content not being displayed as expected, or why certain server-side functions are failing.




Modern dev tools in browsers (above, the "Network" tab in Chrome's DevTools) already capture these requests in real-time.

And—you've guessed it—Marker.io will attach this information automatically with every bug report.

## 9. Source URL

One important, but easy-to-forget item is the source URL.

This will help the developers navigate faster, which saves everyone a lot of time.

 **awesomewebapp.com/pricing**

Pro tip: If you use Marker.io, we'll automatically include this, too!

## 10. Bug severity and priority

By defining the severity or priority of the issue in your bug report, your developer understands how quickly a bug should be fixed.

The severity of your bug can be defined by the level of impact it has on your website or product. Once this has been determined you can label it as:

- Critical
- Major
- Minor
- Trivial
- Enhancement

The priority helps your developer determine which bug they should investigate and fix first. Here you can choose between:

- High
- Medium
- Low

As the bug reporter, you will normally be responsible for identifying the severity and priority.

Pro tip: it can be difficult, as the end-user, to determine bug priority/severity—but if it severely affects functionality and user experience, it's critical!

## 11. Advanced information

Whether you're looking to write the most informative bug report ever or you're looking to score brownie points with your developer, you can also opt to include the following:

- Reporter name (your own)
- Assigned person (the developer, usually)
- Due date

## **The bug report checklist: everything you need to consider**

After lots of personal experience (trial and error—ouch), research, and talking it over with our developers, we came up with a checklist of ten essential points to consider.

1. Title
2. Summary
3. Visual proof
4. Expected vs. actual results
5. Steps to reproduce
6. Environment
7. Console logs (and network requests)
8. Source URL
9. Bug severity and priority
10. Advanced info (optional)