

[Blog](#) › [Tech Topics](#)

# Top 10 JavaScript errors from 1000+ projects (and how to avoid them)

January 24th, 2018 • By Jason Skowronski



To give back to our community of developers, we looked at our database of thousands of projects and found the top 10 errors in JavaScript. We're going to show you what causes them and how to prevent them from happening. If you avoid these "gotchas," it'll make you a better developer.

This site uses cookies to improve your browsing experience. [See Privacy Policy](#)

Accept & close

We focused on the errors most likely to affect you and your users. To do this, we ranked errors by the number of projects experiencing them across different companies. If we looked only at the total number of times each error occurred, then high-volume customers could overwhelm the data set with errors that are not relevant to most readers.

## Here are the top 10 JavaScript errors:

Each error has been shortened for easier readability. Let's dive deeper into each one to determine what can cause it and how you can avoid creating it.

### 1. Uncaught TypeError: Cannot read property

If you're a JavaScript developer, you've probably seen this error more than you care to admit. This one occurs in Chrome when you read a property or call a method on an undefined object. You can test this very easily in the Chrome Developer Console.

```
> var foo;
< undefined
> foo.bar
✖ ▶ Uncaught TypeError: Cannot read property 'bar' of undefined
  at <anonymous>:1:5 VM2523:1
```

This can occur for many reasons, but a common one is improper initialization of state while rendering the UI components. Let's look at an example of how this can occur in a real-world app. We'll pick React, but the same principles of improper initialization also apply to Angular, Vue or any other framework.

```
class Quiz extends Component {
  componentWillMount() {
    axios.get('/thedata').then(res => {
      this.setState({items: res.data});
    });
  }
}
```

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

```

    }
  }

```

There are two important things realize here:

1. A component's state (e.g. `this.state` ) begins life as `undefined` .
2. When you fetch data asynchronously, the component will render at least once before the data is loaded – regardless of whether it's fetched in the constructor, `componentWillMount` or `componentDidMount` . When Quiz first renders, `this.state.items` is undefined. This, in turn, means `ItemList` gets items as undefined, and *you* get an error – "Uncaught TypeError: Cannot read property 'map' of undefined" in the console.

This is easy to fix. The simplest way: Initialize state with reasonable default values in the constructor.

```

class Quiz extends Component {
  // Added this:
  constructor(props) {
    super(props);

    // Assign state itself, and a default value for items
    this.state = {
      items: []
    };
  }
}

```

[Blog](#)
[Guides](#)
[Library](#)


```

componentWillMount() {
  axios.get('/thedata').then(res => {
    this.setState({items: res.data});
  });
}

render() {
  return (
    <ul>
      {this.state.items.map(item =>
        <li key={item.id}>{item.name}</li>
      )}
    </ul>
  );
}

```

This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

Accept & close

clue to either fix or avoid this problem in your app. If not, keep reading because we'll cover more examples for related errors below.

## 2. TypeError: 'undefined' is not an object (evaluating

This is an error that occurs in Safari when you read a property or call a method on an undefined object. You can test this very easily in the Safari Developer Console. This is essentially the same as the above error for Chrome, but Safari uses a different error message.

```
> var testArray=undefined;
  if(testArray.length===0){
    console.log("Array is empty");
  }
✖ Error
  line: 3
  message: "'undefined' is not an object (evaluating 'testArray.length')"
  sourceId: 2054546784
  ▶ __proto__: Error
> |
```

## 3. TypeError: null is not an object (evaluating

This is an error that occurs in Safari when you read a property or call a method on a null object. You can test this very easily in the Safari Developer Console.

```
> var testArray=null;
  if(testArray.length===0){
    console.log("Array is empty");
  }
✖ Error
  line: 3
  message: "'null' is not an object (evaluating 'testArray.length')"
  sourceId: 2056192896
  ▶ __proto__: Error
>
```

This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

Accept & close

```
> undefined === null
< false
> |
```

Please say that again

One way this error might occur in a real world example is if you try using a DOM element in your JavaScript before the element is loaded. That's because the DOM API returns null for object references that are blank.

Any JS code that executes and deals with DOM elements should execute after the DOM elements have been created. JS code is interpreted from top to down as laid out in the HTML. So, if there is a tag before the DOM elements, the JS code within script tag will execute as the browser parses the HTML page. You will get this error if the DOM elements have not been created before loading the script.

In this example, we can resolve the issue by adding an event listener that will notify us when the page is ready. Once the `addEventListener` is fired, the `init()` method can make use of the DOM elements.

```
<script>
  function init() {
    var myButton = document.getElementById("myButton");
    var myTextfield = document.getElementById("myTextfield");
    myButton.onclick = function() {
      var userName = myTextfield.value;
    }
  }
  document.addEventListener('readystatechange', function() {
    if (document.readyState === "complete") {
      init();
    }
  });
</script>
```

```
<form>
  <input type="text" id="myTextfield" placeholder="Type your name" />
  <input type="button" id="myButton" value="Go" />
</form>
```

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

any JavaScript errors (errors that bubble up to the window object, instead of being caught in try-catch) will get reported as simply "Script error" instead of containing useful information. This is a browser security measure intended to prevent passing data across domains that otherwise wouldn't be allowed to communicate.

To get the real error messages, do the following:

## 1. Send the Access-Control-Allow-Origin header

Setting the `Access-Control-Allow-Origin` header to `*` signifies that the resource can be accessed properly from any domain. You can replace `*` with your domain if necessary: for example, `Access-Control-Allow-Origin: www.example.com`. However, handling multiple domains gets tricky, and may not be worth the effort if you're using a CDN due to caching issues that may arise. See more [here](#).

Here are some examples on how to set this header in various environments:

### ***Apache***

In the folders where your JavaScript files will be served from, create an `.htaccess` file with the following contents:

```
Header add Access-Control-Allow-Origin "*"
```

### ***Nginx***

Add the `add_header` directive to the location block that serves your JavaScript files:

```
location ~ ^/assets/ {  
    add_header Access-Control-Allow-Origin *;  
}
```

### ***HAProxy***

Add the following to your asset backend where JavaScript files are served from:

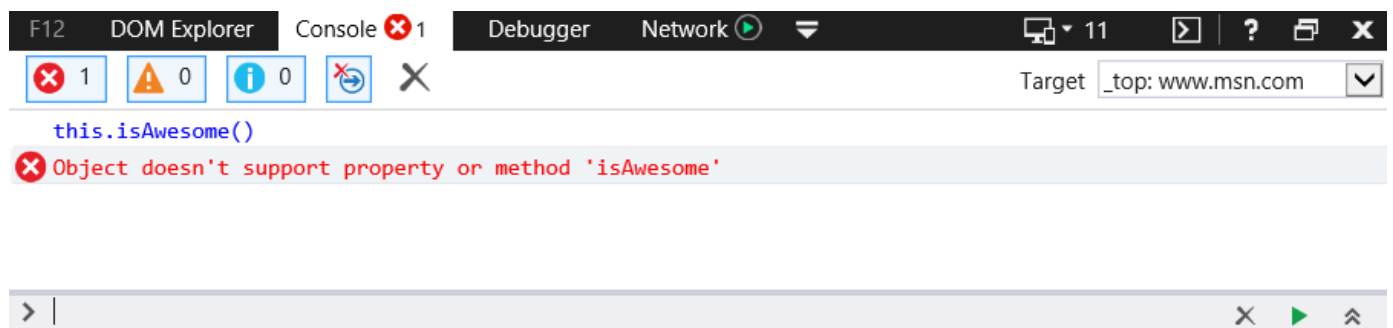
This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

Accept & close

`Origin` header for, set `crossorigin="anonymous"` on the `SCRIPT` tag. Make sure you verify that the header is being sent for the script file before adding the `crossorigin` property on the script tag. In Firefox, if the `crossorigin` attribute is present but the `Access-Control-Allow-Origin` header is not, the script won't be executed.

## 5. TypeError: Object doesn't support property

This is an error that occurs in IE when you call an undefined method. You can test this in the IE Developer Console.



This is equivalent to the error "TypeError: 'undefined' is not a function" in Chrome. Yes, different browsers can have different error messages for the same logical error.

This is a common problem for IE in web applications that employ JavaScript namespacing. When this is the case, the problem 99.9% of the time is IE's inability to bind methods within the current namespace to the `this` keyword. For example, if you have the JS namespace `Rollbar` with the method `isAwesome`. Normally, if you are within the `Rollbar` namespace you can invoke the `isAwesome` method with the following syntax:

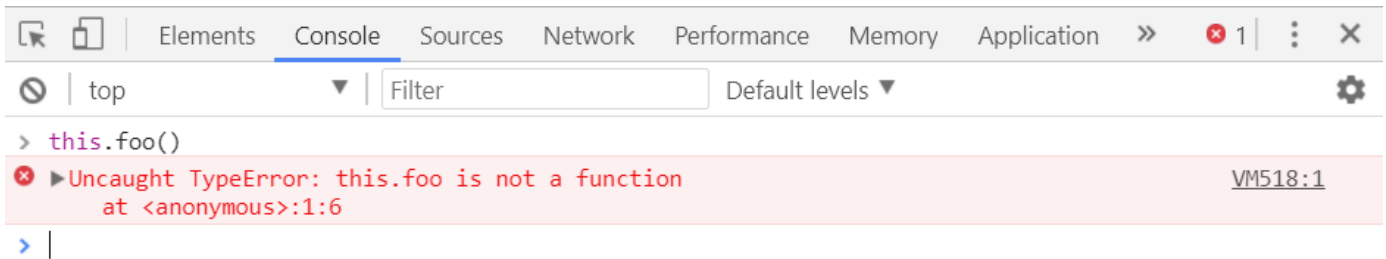
```
this.isAwesome();
```

Chrome, Firefox and Opera will happily accept this syntax. IE, on the other hand, will not. Thus, the safest bet when using JS namespacing is to always prefix with the actual namespace.

This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

Accept & close

This is an error that occurs in Chrome when you call an undefined function. You can test this in the Chrome Developer Console and Mozilla Firefox Developer Console.



As JavaScript coding techniques and design patterns have become increasingly sophisticated over the years, there's been a corresponding increase in the proliferation of self-referencing scopes within callbacks and closures, which are a fairly common source of this/that confusion.

Consider this example code snippet:

```
function clearBoard(){
  alert("Cleared");
}

document.addEventListener("click", function(){
  this.clearBoard(); // what is "this" ?
});
```

If you execute the above code and then click on the page, it results in the following error "Uncaught TypeError: this.clearBoard is not a function". The reason is that the anonymous function being executed is in the context of the document, whereas `clearBoard` is defined on the window.

A traditional, old-browser-compliant solution is to simply save your reference to `this` in a variable that can then be inherited by the closure. For example:

This site uses cookies to improve your browsing experience. See Privacy Policy

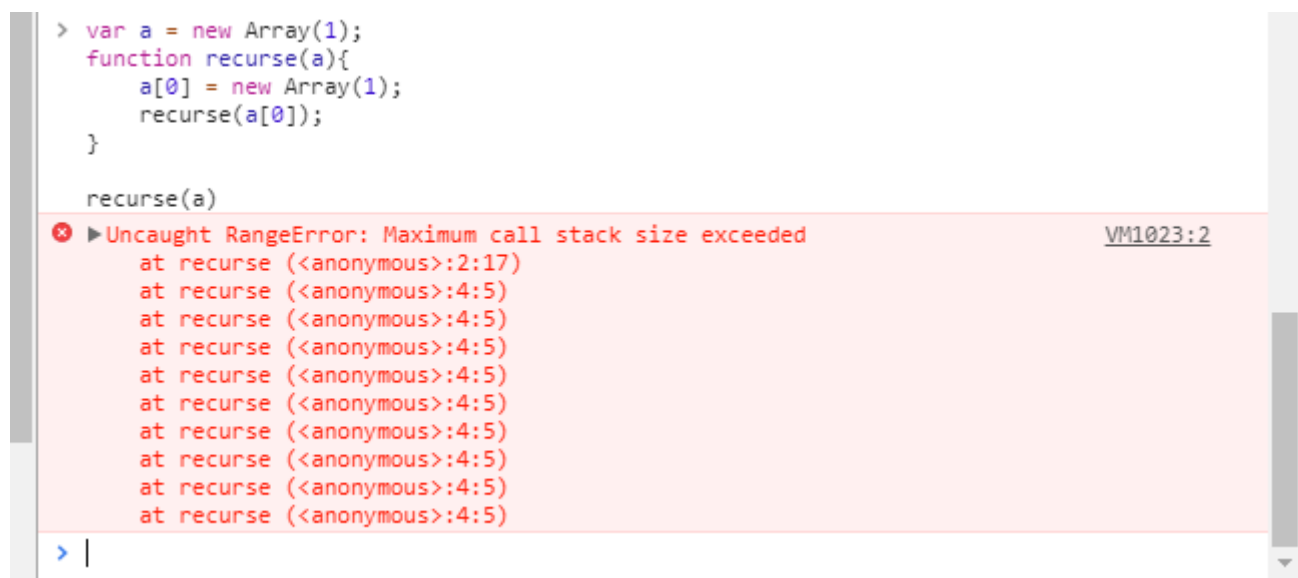
Accept & close



```
document.addEventListener("click",this.clearBoard.bind(this));
```

## 7. Uncaught RangeError

This is an error that occurs in Chrome under a couple of circumstances. One is when you call a recursive function that does not terminate. You can test this in the Chrome Developer Console.



It may also happen if you pass a value to a function that is out of range. Many functions accept only a specific range of numbers for their input values. For example, `Number.toExponential(digits)` and `Number.toFixed(digits)` accept digits from 0 to 100, and `Number.toPrecision(digits)` accepts digits from 1 to 100.

```
var a = new Array(4294967295); //OK
var b = new Array(-1); //range error
```

```
var num = 2.555555;
document.writeln(num.toExponential(4)); //OK
document.writeln(num.toExponential(-2)); //range error
```

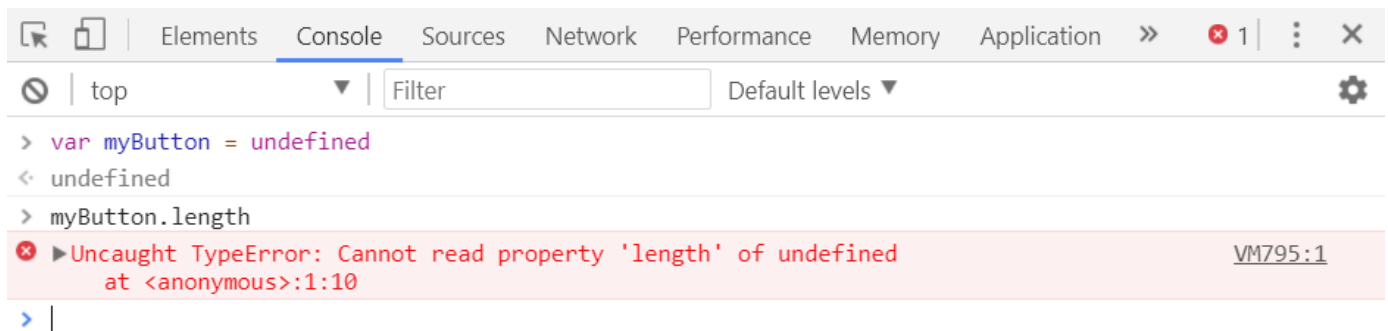
This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

```
document.writeln(num.toPrecision(0)); //range error!
```

## 8. TypeError: Cannot read property 'length'

This is an error that occurs in Chrome because of reading length property for an undefined variable. You can test this in the Chrome Developer Console.



You normally find length defined on an array, but you might run into this error if the array is not initialized or if the variable name is hidden in another context. Let's understand this error with the following example.

```
var testArray= ["Test"];

function testFunction(testArray) {
  for (var i = 0; i < testArray.length; i++) {
    console.log(testArray[i]);
  }
}

testFunction();
```

When you declare a function with parameters, these parameters become local ones. This means that even if you have variables with names `testArray` , parameters with the same names within a function will still be treated as **local**.

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

```
var testArray = ["Test"];

/* Precondition: defined testArray outside of a function */
function testFunction(/* No params */) {
  for (var i = 0; i < testArray.length; i++) {
    console.log(testArray[i]);
  }
}

testFunction();
```

2. Invoke the function passing it the array that we declared:

```
var testArray = ["Test"];

function testFunction(testArray) {
  for (var i = 0; i < testArray.length; i++) {
    console.log(testArray[i]);
  }
}

testFunction(testArray);
```

## 9. Uncaught TypeError: Cannot set property

When we try to access an undefined variable it always returns `undefined` and we cannot get or set any property of `undefined`. In that case, an application will throw "Uncaught TypeError cannot set property of undefined."

For example, in the Chrome browser:

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close



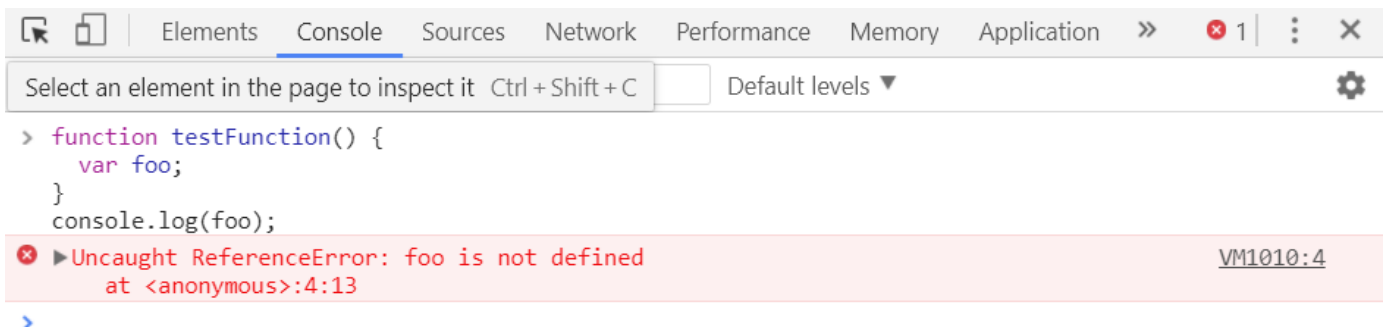
The screenshot shows a JavaScript console with a red error message. The code entered is `var test = undefined` followed by `test.value = 0`. The error message is "Uncaught TypeError: Cannot set property 'value' of undefined" at `<anonymous>:1:12`. The console also shows the variable `test` is `undefined`.

```
> var test = undefined
< undefined
> test.value = 0
✖ ▶ Uncaught TypeError: Cannot set property 'value' of undefined
  at <anonymous>:1:12
VM844:1
```

If the `test` object does not exist, error will throw “Uncaught TypeError cannot set property of undefined.”

## 10. ReferenceError: event is not defined

This error is thrown when you try to access a variable that is undefined or is outside the current scope. You can test it very easily in Chrome browser.



The screenshot shows a Chrome browser console with a red error message. The code entered is a function `testFunction()` that declares a local variable `foo` and then tries to log it. The error message is "Uncaught ReferenceError: foo is not defined" at `<anonymous>:4:13`. The console also shows the function definition.

```
> function testFunction() {
  var foo;
}
console.log(foo);
✖ ▶ Uncaught ReferenceError: foo is not defined
  at <anonymous>:4:13
VM1010:4
```

If you’re getting this error when using the event handling system, make sure you use the event object passed in as a parameter. Older browsers like IE offer a global variable `event`, and Chrome automatically attaches the `event` variable to the handler. Firefox will not automatically add it. Libraries like jQuery attempt to normalize this behavior. Nevertheless, it’s best practice to use the one passed into your event handler function.

```
document.addEventListener("mousemove", function (event) {
  console.log(event);
});
```

This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

Accept & close

JavaScript is expected to have some common error types, and it's best to use guard clauses to check whether objects are undefined before using them.

We hope you learned something new and can avoid errors in the future, or that this guide helped you solve a head scratcher. Nevertheless, even with the best practices, unexpected errors do pop up in production. It's important to have visibility into errors that affect your users, and to have good tools to solve them quickly.

Rollbar gives you visibility to production JavaScript errors and gives you more context to solve them quickly. For example, it offers additional debugging features like **telemetry** which tells you what happened on the user's browser leading up to the error. That's insight you don't have outside of your local developer console. Learn more in Rollbar's full list of **features for JavaScript applications**.

---

If you haven't already, **signup for a 14-day free trial** of Rollbar and let us help you take control of impactful JavaScript errors. :-)

## Get the latest updates delivered to your inbox.

you@email.com

Subscribe

JAVASCRIPT

TOP ERRORS

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close



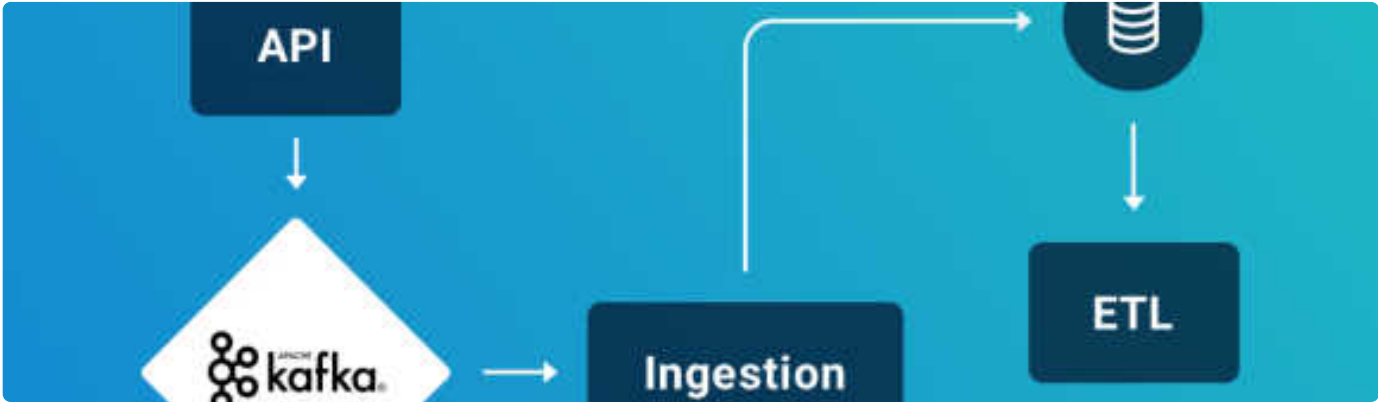
## 5 Ways to Improve Your Dev Team Velocity

June 29, 2020



## Apache Kafka Example: How Rollbar Removed Technical Debt - Part 2

April 07, 2020



## Apache Kafka Example: How Rollbar Removed Technical Debt - Part 1

March 10, 2020

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

# Logging Errors



## Where are JavaScript Errors Logged?

Feb 10, 2020

# JS

# Handling Exceptions



## Exception Handling in JavaScript

Feb 10, 2020

# JS

# Debugging Errors



## How to Debug JavaScript Errors

Feb 10, 2020

This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

Accept & close

JIRA	Laravel
Local Vars	MacOS
Microservices	Monitor
Node.Js	Onboarding
PagerDuty	PHP
Python	Ruby
Salesforce Apex	SDK
Security	Serverless
Slack	Source Maps
Sql	Stacktrace
Team	Telemetry
Triage	User Experience
Versions	

Tutorials

.NET	Angular
AWS Lambda	Behind The Scenes
Debug	Docker
- .. .	- .. ...

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close



Laravel

Magento

Mobile

Monitor

Node.Js

PHP

Python

React

React Native

Ruby

SDK

Serverless

Source Maps

Spring

Sql

Symfony

Telemetry

Triage

Vuejs

Wordpress

Xamarin

Zend

## Tech Topics

APM

Behind The Scenes

Business Impact

Continuous Delivery

Debug

Error Monitoring

Exception Monitoring

Frameworks

Integrations

Java

JavaScript

Kafka

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Subscribe Add Disqus to your siteAdd DisqusAdd

## Product Updates

- |                     |                       |
|---------------------|-----------------------|
| .NET                | API                   |
| APM                 | Atlassian             |
| AWS Lambda          | Azure                 |
| Behind The Scenes   | Bitbucket             |
| Business Impact     | Compliance            |
| Continuous Delivery | Continuous Deployment |
| Debug               | Error Alerts          |
| Error Feed          | Error Monitoring      |

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

Top Errors

Web

# Company Stories

Behind The Scenes

CircleCI

Company Growth

Continuous Delivery

Customer Stories

Customer Support

Error Monitoring

Events

Funding

News

Postmortem

Remote Work

Team

## Try Rollbar for Free

Join 100,000+ developers, improving millions of software experiences

Try for free

Request a demo →

This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

PRODUCT

- Product
- Pricing
- Customers
- Platforms
- Integrations
- Compliance
- Service Status

PLATFORMS

- |            |         |
|------------|---------|
| JavaScript | Angular |
| PHP        | React   |
| Ruby       | Laravel |
| Python     | Node    |
| iOS        | Rails   |
| Java       | Django  |
| .NET       | More... |

DOCUMENTATION

- Docs Overview
- Setting up Rollbar
- Notifications
- Deploy Tracking
- Telemetry
- Security & Compliance
- API

COMPANY

- About Us
- Events
- Careers
- Media
- Contact Us

RESOURCES

- Rollbar Product Blog
- Learn How Rollbar Complements New Relic
- Using Rollbar with Atlassian
- Using Rollbar with GitHub
- Pre-production Error Monitoring
- Low-Risk Continuous Delivery
- Salesforce Apex Error Logging



This site uses cookies to improve your browsing experience. See Privacy Policy

Accept & close

This site uses cookies to improve your browsing experience. See [Privacy Policy](#)

**Accept & close**