BEST PRACTICE SERIES

Frontend Monitoring

DATADOG

BEST PRACTICE SERIES

Frontend Monitoring

Best practices for modern frontend monitoring

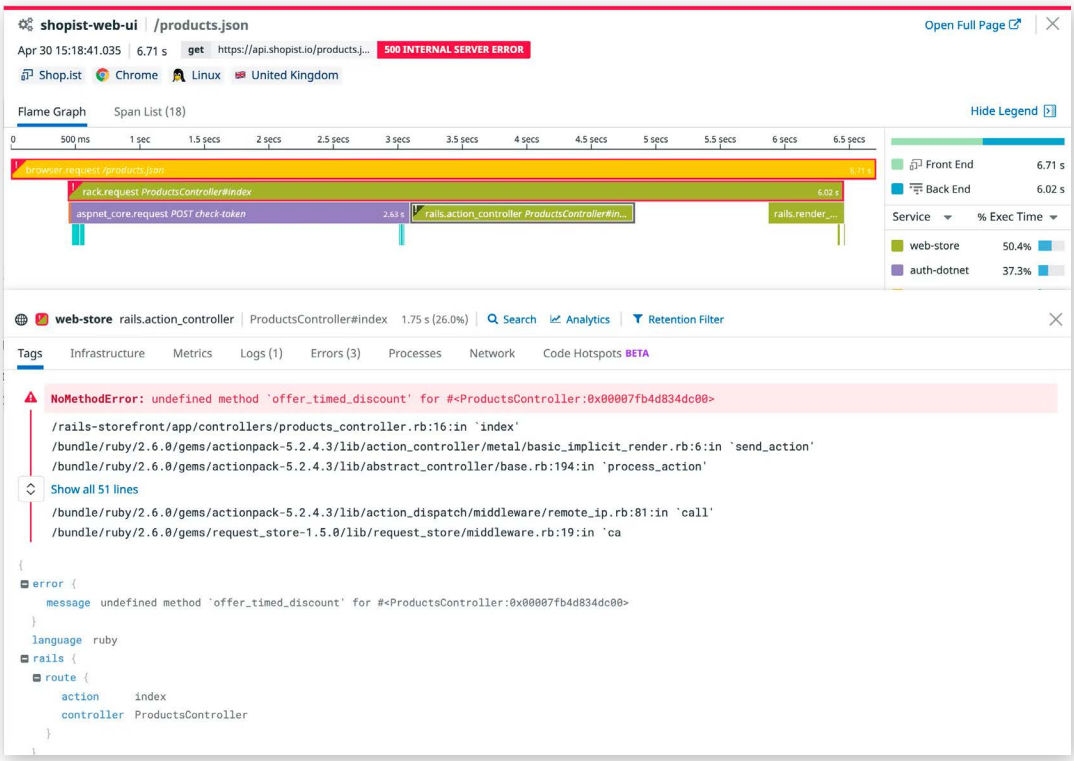
When building out your web application, it's important to decide which architectural approach is best suited for ensuring its performance and reliability. While multiple-page apps (MPAs) are the traditional implementation, single-page applications (SPAs) provide some significant benefits over MPAs. For JavaScript developers using frameworks like React or Vue, they offer flexibility in moving application logic to the frontend, reducing the need for complex backend operations. For users, SPAs can provide a smooth experience with a highly interactive UI and fewer page loads. But, with increased sophistication, there are some tradeoffs. For example, users could be accessing your site on older or less powerful devices than your developers can test on, which may mean that the additional complexity is the user's burden. Developers of SPAs need to monitor end-user experiences in order to ensure their sites run smoothly on all devices.

However, frontend monitoring concepts and tools are still evolving to keep up with JavaScript and SPA frameworks. Traditional methods of monitoring frontend performance are based on measuring the timing of completed page loads. In the case of SPAs, which only have a single page, this method only accounts for the first few seconds of the user experience. As the user navigates an SPA, it will re-render page components to reflect the new data state instead of loading a new document. In order to accurately monitor the performance of your application, you need to be able to track dynamic page elements like animations, API calls, and rendering lifecycles that can interrupt user experience if there are slowdowns.

When it comes to frontend monitoring, we recommend a user-centric approach by listening to browser events and tracking user interactions over time. Google and the W3C Web Performance Working Group have worked on defining new APIs that take observation of user interactions as the basis of site performance. With these APIs, you can build visibility into your app by breaking down interactive elements and timing user actions as finely as possible

In this article, we'll specifically look at how to measure three aspects of single-page apps that can help you optimize your users' experience:

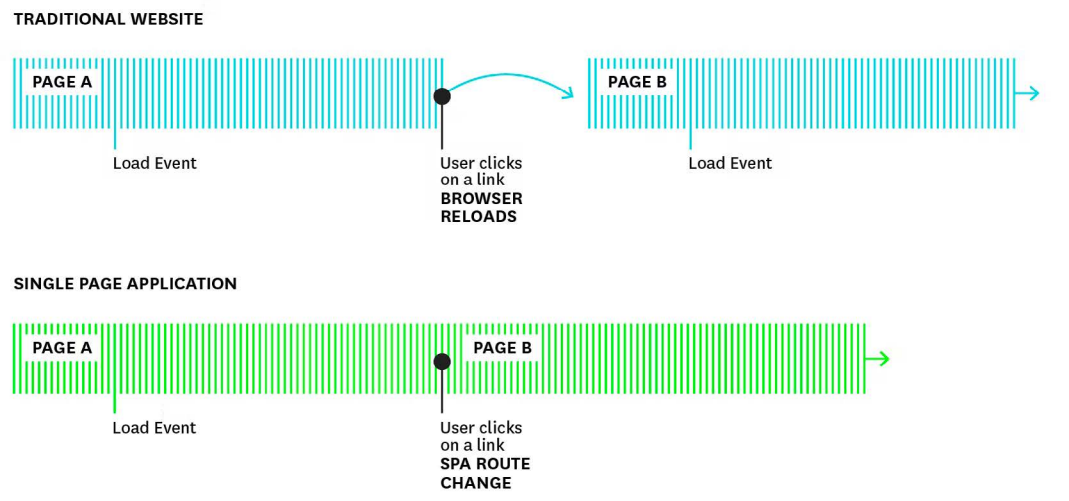
- Route changes, to optimize page transitions
- User interactions, to ensure a smooth user experience
- Errors, to catch and debug issues in the browser



Route changes

Fluid transitions between different parts of a web application are crucial for good end-user experience. Let's say you maintain an e-commerce single-page application that contains a promotional landing page. On the landing page is a large hero image advertising a product. Users can visit this page either via a direct link, or by clicking through the main site. In order to ensure smooth page transitions, you decide the hero image should load in two seconds or less. Beyond that, users could become frustrated and leave the site.

One common way to measure site transitions is by tracking the Largest Contentful Paint (LCP). This estimates when a page has finished loading based on how long it took for the largest `<img>`, `<video>`, or `<svg>` element to become visible. MPAs will reload the browser when a link is clicked, but because SPAs modify the browser route and re-render the site components, the largest element on the previous page might not reload on the new one. So, measuring loading times requires a way to detect changes in the browser that don't depend on standard loading events. The illustration below shows the sequence of browser loading events on a traditional website versus an SPA:



In this case, instead of page load events, we recommend listening to route changes and using those to calculate performance metrics.

LISTEN TO ROUTE CHANGES

In order to track route changes, you need to instrument your code to watch for different browser events than a full page load. The User Timing API provides functions that make it easy to define your own loading indicators so that you can precisely measure page and resource load times.

First, using the browser History API, listen for the popstate event, which is fired when the browser route changes. (Be careful to take into consideration instances where it might not indicate a whole new page.Then, use the User Timing API to set a `window.performance.mark()` to indicate the route change has begun.

route-change-mark.js

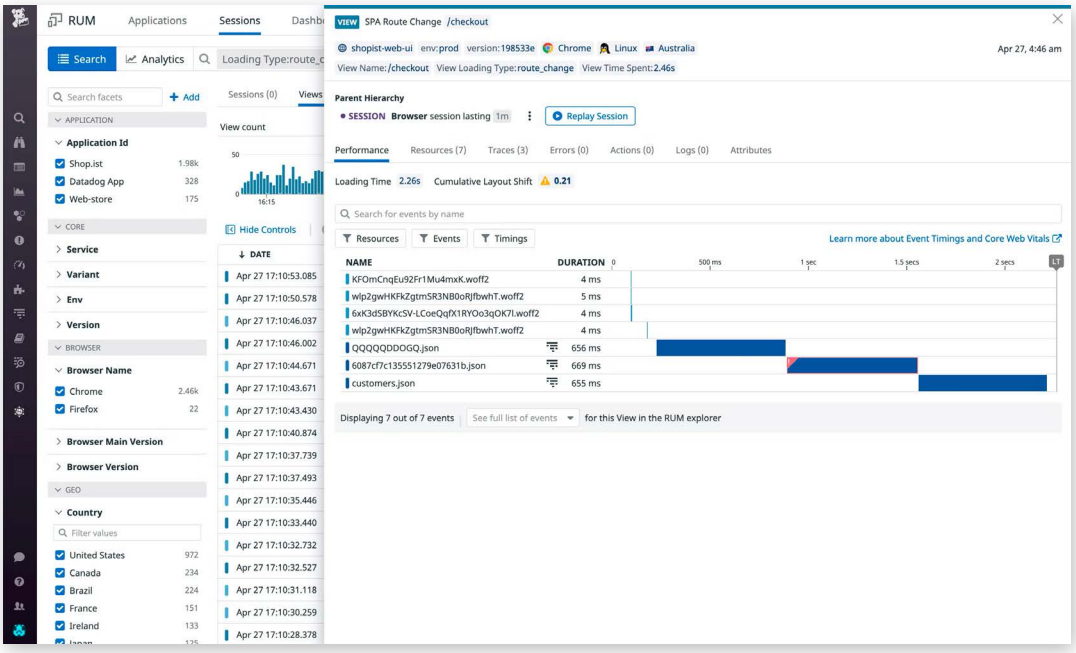
```
window.onpopstate = function(e) {
   window.performance.mark('route_change_start');
}

...after requests finish and components render

window.performance.measure('route_change','route_change_start')
```

Once the component of the new view that you select as your indicator of readiness has been rendered (in this case, the hero image has appeared), you can pass the mark to the window.performance.measure function, which creates a finish mark and calculates the difference between the two. This makes it easy to log the loading time of your application's route changes, which you can then forward to a logging service, such as Datadog.

When timing each route change event, it's helpful to also track the loading time of individual resources to see where slowdowns are occuring. Below is a waterfall visualization of a route change that shows how long it takes to load some web fonts and JSON files. We can see that the overall view took 2.26 seconds to fully load, and iIt's easy to see exactly which files are slowing down the site and which one encountered an error while loading.



AUTOMATE TO SAVE TIME

Manually tracking route changes can become cumbersome if you're adding markers for each page transition. Another way to time page transitions is to create a loop that triggers after a route change has been detected. For example, after creating the initial performance marker, you can instrument your application to check every 100ms to see if there have been any network requests or DOM mutations. If any are initiated or completed during that time, your application will wait another 100ms. Otherwise, it takes the previous 100ms mark as your loading time. Datadog uses a similar method to track page loads for SPA route changes.

User Interactions

Tracking the duration of key user interactions across your application can reveal slowdowns. To optimize user flow, it's important to collect as much fine-grained information as possible.

Using the same e-commerce site example, when a user adds an item to their cart, the action triggers a modal popup with some suggested products. This is a key interaction to monitor because it triggers a lot of activity in the browser, including a series of API calls, updating the data state, and then re-rendering the components. Latency during any of these tasks can quickly add up, especially if the user is running the site on an older mobile device or computer. If the popup takes too long to load or respond to input the user will feel blocked from their goal (buying something), and the site potentially loses a customer.

As with route changes, you can add event listeners to key UI elements, such as button clicks, scrolls, and key presses, which trigger state changes. This allows you to easily track load times for important user interactions and find ones that take the longest. Again, 100ms has been proven to be the threshold above which users lose the feeling that interactions are natural and fluid.

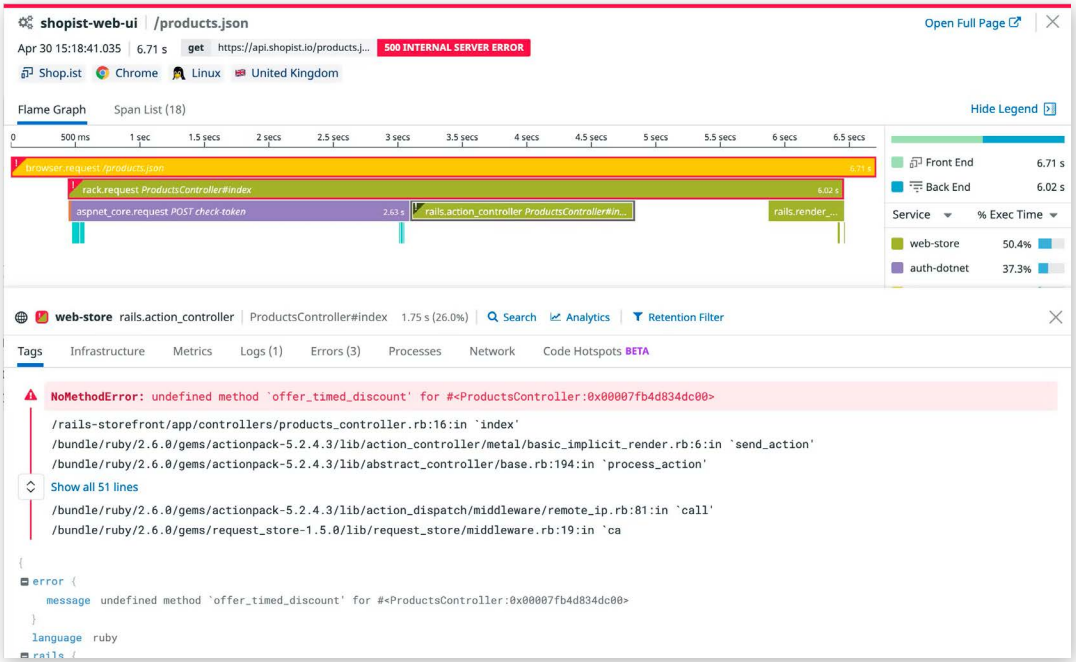
In addition to using event listeners to track load times, it's also important to be able to visualize traffic across each key step in your user journey. With funnel analysis, a new feature available for Datadog RUM customers, you can leverage the user session data captured by Datadog to understand if users are successfully completing key workflows for your business's health—like e-commerce checkouts, account creation, or email signups. Once you select the sequence of views and actions you want to analyze, the resulting funnel graph shows what percentage of users move from each step to the next one.

You can enhance these insights with Datadog Session Replay, which offers a direct view into how users are actually navigating through your application. Session Replay's video-like recreations of real user journeys make it easier to reproduce bugs and identify patterns in user behavior. The ability to understand exactly how your users interact with your website saves you time, minimizes guesswork, and helps focus your frontend optimization efforts.

IMPORTANCE OF END-TO-END MONITORING

Key user interactions within an SPA often depend on multiple behind-the-scenes jobs, such as API calls, data state changes, and component re-rendering. This means that monitoring the performance of these backend processes is crucial when determining the cause of any latency in the frontend. For instance, slowdowns in your suggested product modal may be the result of an API timeout, which wouldn't necessarily be revealed by monitoring only the loading time of the modal itself. Having visibility into different layers of your stack can help you pinpoint where code or database issues are causing slowdowns on the frontend.

For example, below we see a flame graph of a request initiated from the frontend (to fetch /products.json). We can see a breakdown of all of the network requests and backend tasks this request triggered. Each span appears with its duration and detailed context from the execution, including any errors that may have occurred. With a clear timeline and hierarchy of API actions, you can tell right away the problem was a NoMethodError originating in the controller. Without this visibility into how requests propagate across our stack, it is much more difficult to isolate where the causes are of any latency in the user experience.



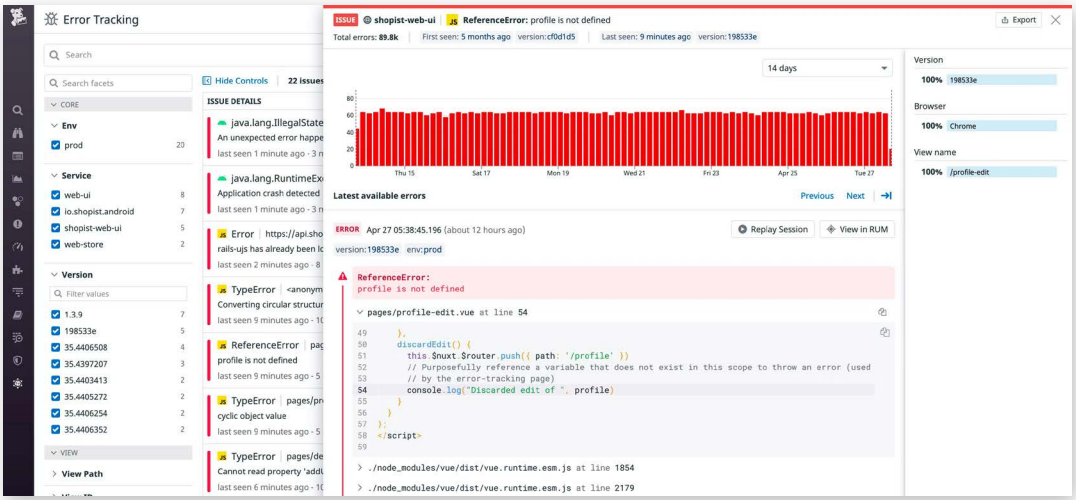
Watching for Long Tasks on the frontend can also help surface slowdowns impacting user experience and scripts requiring optimization. A Long Task is defined as a section of JavaScript code blocking the main UI thread for at least 50ms. You can catch these either on the code level, or in your browser profiler for quick visualization and debugging.

Error tracking

Since SPAs run more complex code on the user's device, you won't find their browser errors unless you instrument your code to log them. If you don't, you may only find out about errors when support tickets come in or revenue drops. Catching and logging application errors will help you spot issues before this happens. For example, if the "checkout" button on your e-commerce site is unable to connect to the backend API, it would cause an error in the browser. When a user clicks the button and nothing happens for them, they might become frustrated and leave the site. Without tracking user browser errors, you would've only been alerted to the problem from a rise in the number of abandoned carts.

You can do this by using a `GlobalEventHandler` like window.onError to catch and log error events. When logging errors, you should collect as much information as possible about the user's browser, OS, application version, etc. This provides more context around the problem's impact and possible cause when you're debugging.

In order for error tracking to be effective, you need to be able to separate signal from noise. For example, third-party libraries, browser extensions, and other dependencies may generate errors unrelated to your site's performance. It can be challenging to filter through them for actionable information. Using a third-party monitoring solution to aggregate similar error logs can help you analyze them and surface critical issues. Below is an example Datadog's Error Tracking, which enables you to more easily triage errors from across your stack.



TACKLE ERRORS BEFORE THEY HAPPEN

Creating and maintaining an end-to-end testing workflow to test your application in real-world conditions can help you detect browser errors before they affect your users. Synthetic monitoring works by simulating API requests and site interactions to test complex user journeys, like adding items to a cart or signing up for a service.

Third-party tools like Datadog Synthetic Monitoring allow you to automate test creation and execution, so you can simulate site behavior under a variety of conditions, like a user's geographic location or the type of device they're using. This way, you can integrate frontend behavior testing as part of your CI workflow and catch faulty code before it interrupts the user experience.

Start monitoring your SPAs

As frontend frameworks and SPAs continue to grow in complexity, it can be difficult to measure performance when the bulk of the rendering is done on the user's device. That's why we recommend thinking of frontend monitoring in terms of tracking the user journey. By timing user interactions and resource loading, as well as staying on top of in-browser errors, you can create visibility and collect actionable insights to optimize your single-page application.

A monitoring solution like Datadog can help you get deep visibility into the frontend performance of your applications along with the rest of your stack. Datadog Real User Monitoring (RUM) automatically tracks latency for user actions and collects key metadata such as the user's browser, OS, and device, as well as their geolocation. With Session Replay, you can watch video-like recreations of real user sessions to identify patterns in user behavior, while the funnel analysis helps you understand usage and dropoff trends from aggregated session data. Datadog Synthetic Monitoring enables you to create browser tests that simulate real-world conditions to test key user journeys within your frontend application. Error Tracking collects and aggregates JavaScript errors for alerting and troubleshooting. And, together with Datadog APM, you can correlate frontend and backend performance of requests as they propagate across your stack, all in a single pane of glass.

Get started monitoring your SPAs with Datadog today. If you're not signed up with Datadog yet, you can start your free trial.


DATADOG