Front-End Performance: Challenges and How to Solve Them

Front-end performance testing can be challenging. There's no question about that. And it should be a major part of the testing regime for any browser-based application or service. There's no question about that, either.

The real questions are: How do you meet these challenges? And how can you successfully incorporate your approach into your test regime? In this post, we'll take a closer look at those questions, and at ways to meet the front-end testing challenge. But first, some background on front-end performance testing — what it is, and why it's important.

Testing the Front End

Front-end performance testing is basically just what the name implies: testing your site's performance in the end-user's browser. It is relatively easy to test server-side performance, since you are typically able to closely control the server environment, monitor it in-depth, and subject it to the conditions that you want to test with some precision. But server-side testing only tells you about the server's behavior — how it responds to user requests, how quickly it supplies content, and how it manages data retrieval and computation on its end. It doesn't give you a clear or even accurate picture of the end-user experience. For that, you need front-end performance testing.

Simple HTML ...

There was a time (in the rather distant past) when front-end performance was simple almost to the point of being trivial: the browser loaded a page consisting largely of HTML, with some graphics, and possibly some lightweight scripting. The heavy lifting was mostly or entirely at the server end, and the greatest threat to performance was likely to be the connection speed, followed by server overload. The browser spent much of its time with a fully loaded page of HTML, just waiting for graphics and other content.

… But Simple No More

But that was then, and this is now. Client-side scripting technologies such as Ajax are well-established, and they, along with CSS, have long since effectively redefined the web page — a typical page is now likely to be a collection of scripting elements and references to external resources contained in an HTML framework. When the browser loads the HTML, its job has just begun. Performance depends almost entirely on everything that happens after that.

Elements of Front-End Performance

As seen from the user's point of view, front-end performance is likely to consist largely of the following elements:

* How quickly basic page features load:  
  + Visible text
  + Graphics
  + Formatting and layout (CSS)
  + Functional elements (buttons, links, forms, etc.)
* How quickly functional elements become responsive (at all) to user actions
* How quickly functional elements are able to carry out user requests
* How long it takes for the entire page and all of its functionality to load.

In a real-world setting, not all of these elements are likely to carry equal weight in the minds of most users, and their importance is likely to vary, depending on the purpose of the page. In a page meant to display graphics and do little else, for example, the speed at which multiple images load may be more important than any other consideration.

Basic Performance Priorities

For most pages associated with web-based applications, you can typically assign the following rough priorities to key elements:

1. Basic text and layout. The visible framework of the page should load quickly, with key text elements placed in the correct location.
2. Functional elements. Ideally, these should be visible, responsive, and fully functional as soon as possible.
3. Formatting and images. In a page where graphics are not crucial, CSS and graphics can load after functional elements.

Note, however, that all of these processes should be asynchronous. To the degree that it is possible, no element should have to wait until another element has loaded in order to become visible to the user.

Challenge: Matching the Real-World Environment

In many ways, this is the most fundamental challenge of front-end performance testing. Your users access your site under real-world conditions, and not in an idealized, controlled test environment. That's easy enough to see. But each user's real-world environment is different, in both large and small ways. You can't duplicate the exact environment in which every user operates, so how do you set up test cases that accurately reflect real-world conditions?

You can't cover the ground completely, but you can identify key environmental factors which may affect page performance. These typically include available memory and CPU performance on the client system, as well as connection speed. But while you can't duplicate all possible real-world conditions, you should be able to do a good job of modeling a user with an overloaded mobile device trying to access your site using the not-so-great WiFi at the neighborhood coffee house, or a home user accessing your shopping cart and checkout pages while watching high-bitrate streaming video on a low-resource laptop.

There's a good chance that a handful of use cases will accurately reflect the real-world conditions of the majority of your users — and that the adjustments you make to improve performance under those conditions will also apply to many of the less-common use cases.

Challenge: Which Elements Are Important, and When Do They Load?

The basic priorities that we listed earlier are just that — basic. You need to have a clear understanding of which elements on your key pages should have the highest priority, and ideally, you should be able to develop a clear picture of when they load to the point where they are functional (or adequately visible). Measuring the speed at which specific individual elements load may occasionally require some manual testing, but a surprising amount of performance in these areas can be tested automatically by looking at such metrics as First Meaningful Paint and Time to Interactive.

The key to adequate front-end performance testing of specific elements is to understand (and list) loading priorities on a page-by-page basis before you map out your test regime. It isn't simply a matter of using testing time and resources efficiently — It will also make it easier for you to avoid the mistake of optimizing loading time for less-important elements at the expense of those which should be given a higher priority.

Challenge: Is It Front-End or Server-Side?

It isn't always easy to tell whether a problem is really on the front end, or if it's a server issue. Failure to display a specific element could be the result of something going wrong at the front end, or it could simply be a case of the server or CDN failing to deliver content. The most effective front-end performance metrics for sorting out client issues from server problems are those which mark early interactions and those which indicate completed actions.

Time to First Byte (TTFB), for example, is the time between the initial client-side request for a page and the arrival of the page's first byte at the browser. TTFB is largely a measure of factors outside of the browser environment — typically, the connection speed and the server response time. As such, it can be used as a benchmark for separating server/connectivity speed from strictly front-end issues.

If you know the actual time at which the first byte arrived, you can also compare it with metrics indicating the completion of key actions, such as DOM Content Loaded and Time to Interactive. A short TTFB followed by an excessively long time until DOM or interactive content is loaded, or a long gap between First Paint and First Meaningful Paint, is a reasonably good indicator that the delays are likely on the front end, rather than with the server or the Internet connection.

A front-end performance testing tool which makes it easy to capture and analyze these metrics will speed up the process of front-end optimization, and give you a clear ongoing picture of how your site performs under real-world conditions.