sitepoin

FRONT-END PERFORMANC

Because every wasted second costs you money

Front-end Performance

- **Product Manager:** Simon Mackie
- **English Editor:** Ralph Mason
- **Technical Editor:** Darin Dimitrov
- **Cover Designer:** Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066
Web: www.sitepoint.com
Email: books@sitepoint.com

ISBN: 978-0-9943469-6-4 (print)

ISBN: 978-0-9943470-2-2 (ebook)

Printed and bound in the United States of America

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our blogs, books, newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, Ruby, mobile development, design, and more.

Table of Contents

Chapter

1

Which Browsers Should Your Website Support?

Craig Buckler

This chapter is part of a series created in partnership with SiteGround. Thank you for supporting the partners who make SitePoint possible.

The question *"which browsers should my website/app support?"* is often raised by clients and developers. The simple answer is a list of the top N mainstream applications. But has that policy become irrelevant?

What are the Most-used Browsers?

The top ten desktop browsers according to StatCounter for May 2017 were:

1. Chrome --- 59.37% market share
2. Firefox --- 12.76%
3. Safari --- 10.55%
4. IE --- 8.32%
5. Edge --- 3.42%
6. Opera --- 1.99%
7. Android (tablet) --- 1.24%
8. Yandex Browser --- 0.48%
9. UC Browser --- 0.41%
10. Coc Coc --- 0.33%

Mobile now accounts for 54.25% of all web use so we also need to examine the top ten phone browsers:

1. Chrome --- 49.23%
2. Safari --- 17.73%

3. UC Browser --- 15.89%
4. Samsung Internet --- 6.58%
5. Opera --- 5.03%
6. Android --- 3.75%
7. IEMobile --- 0.68%
8. BlackBerry --- 0.26%
9. Edge --- 0.15%
10. Nokia --- 0.12%

The worldwide statistics don't tell the whole story:

- Patterns vary significantly across regions. For example, Yandex is the second most-used Russian browser (12.7% share). Sogou is the third most-used browser in China (6.5%). Opera Mobile/Mini has a 28% share in Africa.
- New browser releases appear regularly. Chrome, Firefox and Opera receive updates every six weeks; it would be impractical to check versions going back more than a few months.
- The same browsers can work differently across devices and operating systems. Chrome is available for various editions of Windows, macOS, Linux, Android, iOS and ChromeOS, but it's not the same application everywhere.
- There is an exceedingly long tail of old and new, weird and wonderful browsers on a range of devices including games consoles, ebook readers and smart TVs.
- Your site's analytics will never match global statistics.

Are Browsers So Different?

Despite the organic variety of applications, all browsers have the same goal: *to render web pages*. They achieve this with a rendering engine and there is some cross-pollination:

1. Webkit is used in Safari on macOS and iOS.
2. Blink is a fork of Webkit now used in Chrome, Opera, Vivaldi and Brave.
3. Gecko is used in Firefox.
4. Trident is used in Internet Explorer.
5. EdgeHTML is an update of Trident used in Edge.

The majority of browsers use one of these engines. They're different projects with diverse teams but the companies (mostly) collaborate via the W3C to ensure new technologies are adopted by everyone in the same way. Browsers are closer than they've ever been, and modern smartphone applications are a match for their desktop counterparts. However, no two browsers render in quite the same way. The majority of differences are subtle, but they become more pronounced as you move toward cutting-edge technologies. A particular feature may be fully implemented in one browser, partially implemented in another, and non-existent elsewhere.

Can My Site Work in Every Browser?

Yes. Techniques such as progressive enhancement (PE) establish a baseline (perhaps HTML only) then enhance with CSS and JavaScript when support is available. Recent browsers get a modern layout, animated effects and interactive widgets. Ancient browsers may get unstyled HTML only. Everything else gets something in between. PE works well for content sites and apps with basic form-based functionality. It becomes less practical as you move toward applications with rich custom interfaces. Your new collaborative video editing app is unlikely to work in the decade-old IE7. It may not work on a small screen device over a 3G network. Perhaps it's possible to provide an alternative interface but the result could be a separate, clunky application few would want to use. The cost would be prohibitive given the size of the legacy browser user base.

Site Owner Recommendations

Site owners should appreciate the following fundamentals and constraints of the web. **The web is not print!** Your site/app will not look identical everywhere. Each device has a different OS, browser, screen size, capabilities etc. **Functionality can differ** Your site can work for everyone but experiences and facilities will vary. Even something as basic as a date entry field can has a diverse range of possibilities but, ideally, the core application will remain operable. **Assess your project** Be realistic. Is this a content site, a simple app, a desktop-like application, a fast-action game etc. Establish a base level of browser compatibility. For example, it must work on most two-year-old browsers with a screen width of 600 pixels over a fast Wi-Fi connection. **Assess your audience** Don't rely on global browser statistics. Who are the primary users? Are they IT novices or highly technical? Is it individuals, small companies or government organisations? Do they sit at a desk or are they on the move? No application applies to everyone --- concentrate on the core users first. Examine the analytics of your existing system where possible but appreciate the underlying data. If your app doesn't work in Opera Mini, you're unlikely to have Opera Mini users. Have you blocked a significant proportion of your market? **Change happens** It's amazing that a web page coded twenty years ago works today. It won't necessarily be pretty or usable but browsers remain backward compatible. *(Mostly. The `<blink>` tag can stay dead!)* However, technology evolves. The more complex your site or application, the more likely it will require ongoing maintenance.

Web Developer Recommendations

With a little care it's possible to support a huge variety of browsers. **Embrace the web!** The web is a device-agnostic platform. Content and simpler interfaces can work everywhere: a modern laptop, a feature phone, a games console, IE6, etc. Learn the basics of progressive enhancement. Even if you choose not to adopt it for your full application, there will be pockets of functionality where it becomes invaluable. **Adopt Defensive Development Techniques** Consider the problem before reaching for the nearest pre-written module, library or framework. Understand the consequences of that technology before you start. Frameworks

should provide a browser support list because they have been tested in limited number of applications. Learn about browser limits and quirks. For example, if you're considering an SVG chart, be aware that it can look odd in IE9 to 11 and fail in IE8 and below. That doesn't mean it's a binary choice of rejecting SVGs or abandoning IE support. There are always compromises which do not incur significant development. For example:

- accept SVG rendering is weird but it remains usable
- only show a table of data in IE, or
- provide an SVG download which IE users can open elsewhere.

Test early and test often You cannot possibly test every device, but developing for a single browser is futile. Continually test your project in a variety of applications. Leaving testing to the end will have catastrophic consequences. It's easy for us to blame tools and browser inadequacies, but the majority of issues can be rectified during the development process if they're spotted early. That's not to say everything must work identically in every browser every time. Feature regressions are inevitable. For example:

- Progressive Web Apps do not work offline on iPhones and iPads --- but online operation is fine.
- CSS Grid is not supported in IE --- but float, flexbox or full-width block fallbacks should be acceptable.
- The desktop edition of Firefox does not show a calendar for date fields --- but users can still enter one.

Install a selection of browsers on your development PC. Mac and Linux users can obtain Microsoft Edge and IE testing tools at developer.microsoft.com/microsoft-edge/. It's more difficult for Windows and Linux users to test Safari; online test services such as BrowserStack are the easiest option. Modern browsers have excellent mobile emulation facilities, but use a few real devices to appreciate touch control and performance on slower hardware and networks.
Use HTTPS on your end The web is gradually making HTTPS the preferred protocol, and this trend is going to continue. Google Chrome is even starting to

indicate that non-HTTPS sites as insecure, which is a good reason for you to configure your site to use HTTPS. Our web hosting partner, SiteGround, for example, made it easy for their clients to make the move to HTTPS. To do that, they automated the installation of Let's Encrypt SSL certificates for all new WordPress accounts, and for existing ones, they made the switch to HTTPS possible with just a click.

You Haven't Answered the Question!

The question *"which browsers should you support?"* has become too restrictive. Presume your answer was just "Chrome":

- *which devices and OS is it running on?*
- *what range of screen sizes will be supported?*
- *which version are you referring to? The latest? Chrome 10 and above?*
- *what happens when a new version of Chrome is released?*
- *what will happen in other browsers when Chrome effectively becomes your application's runtime?*

Providing a browser support list has become impractical for client-facing projects. Perhaps the best answer is: *"we'll develop your project according to presumed demographics then test as in many devices, OSes, browsers, and versions as possible according to budget and time constraints"*. Even then, you'll miss that aging Blackberry the CEO insists on using. Develop for the web --- *not browsers*.

Chapter

2

Are Your WordPress Themes Flexible or Fast?

Maria Antonietta Perna

This article is part of a series created in partnership with SiteGround. Thank you for supporting the partners who make SitePoint possible.

If you've developed WordPress themes for mass distribution, you might have come across the problem of finding the perfect balance between building performant themes and building feature-rich, media-heavy products for your customers.

Let's look closer into what the tension might be all about and how you can find ways to compromise between creating fast-loading themes and giving users the flexibility and easy customization options they love and expect.

Are Flexibility and Performance at Odds When Coding WordPress Themes?

I will start by saying that my discussion is not going to be about performance in relation to an entire WordPress website, which might include a number of different factors like finding a great hosting provider, implementing caching mechanisms, leveraging both back-end and front-end techniques, etc.

Also, the topic is not about the performance of WordPress themes you code from scratch either for your own use or for a specific client. In these particular cases you tailor your themes to the specific needs of yourself or your individual client, which should make performance optimizations easy to accommodate.

Rather, my focus will be on WordPress themes you code for the general public, to be distributed on WordPress.org or an online marketplace. The scenario is one where you, as the theme developer, have no control over how your theme is going to be used and customized.

But why could coding for performance clash with coding themes for the public?

On the whole, performance requires you to:

- stick to simple designs
- include a limited number of features into your themes (more features are likely to require more processing power and resources, all of which impacts on theme performance)
- add the minimum number of page templates for a theme to function (fewer templates require fewer resources, which is good for performance)
- perform as few database queries as possible (querying the database takes time)
- limit the number and size of images and other media, which are notoriously heavy files
- minimize the number of HTTP requests (each round trip to the server and back takes some time with obvious negative consequences on performance).

On the other hand, a considerable number of your theme's users are likely more entrepreneurial than tech minded. Therefore, what they might be looking for is a product they can turn into pretty much anything without so much as one line of code, with lots of functionality out of the box, breathtaking photo and video assets, super delightful parallax and other animation effects, and such like.

To give theme users all the flexibility they would like could come with some performance costs. But let's go into a few specific points that illustrate how this can happen and how you can find some middle ground for a successful outcome.

Themes are for appearance, plugins for functionality

Although a number of reputable WordPress experts have been throwing some well-argued criticisms against the so-called *kitchen sink themes* --- that is, those multipurpose themes that can become anything to anybody by offering any functionality under the sun --- the demand for them is still going strong.

Themes that give users the ability to easily add social media buttons, SEO features, contact forms, price tables, etc., while attracting a lot of attention from buyers, don't come without some serious drawbacks.

In particular, this kind of WordPress theme comes tightly coupled with plugins specifically built for it, or even integrated directly into the theme. This practice is widely frowned upon for the following reasons:

- Some of the plugins that come bundled with a theme can have vulnerabilities that put theme users at risk. If the theme doesn't come with specific plugins installed it's less open to such risks
- A super important drawback of tight integration between themes and plugins is what Ren Ventura identifies as *theme lock*. Here's his explanation of this problem:

  > Theme lock occurs when a WordPress user cannot change his or her theme without gutting most of the site's functionality. Once the theme is deactivated, it deactivates things like shortcodes and custom post types that were registered by the theme. Without these features that the user has heavily incorporated into the site, things fall apart.

  As a consequence, when you change theme, a lot of the stuff you thought it was part of your website content disappears with the old theme. For instance, if your theme offered the ability to add testimonials to your website, once you change theme all the content related to your testimonials will be gone. Not nice.

- And of course, bloat and hence performace costs. Let's say you don't need a testimonials section on your website. Yet, all the code that makes that functionality work in the theme is still there: it takes up space on your server, which ultimately costs money.

A great maxim the Theme Review Team (i.e., the volunteers who review themes submitted on the WordPress.org themes repo) rightly enforces is: **keep functionality separate from appearance. Plugins deal with functionality, themes with appearance.** Getting rid of all the complication will improve theme performance and at the same time make it easier for users to install and

configure WordPress themes.

Your users don't need tons of theme options (but they might not know this)

Until not long ago, WordPress themes included rather complicated theme option pages to enable users to make all sorts of modifications with a button click.

These days, thanks to a momentous decision taken by the Theme Review Team on WordPress.org in 2015, most themes offer theme options using the WordPress Customizer, which makes possible live previewing the changes as they're being made by users.

Unfortunately, the kitchen sink mentality that ruled the old theme option pages has started to migrate to the Customizer, which you can now see as also starting to get filled up with all sorts of settings.

Although non technical customers love theme options to make changes to their themes, sometimes too many options available bring more problems than they seemingly solve:

- Too many options can paralyze users who are not too familiar with core principles of website design.
- It can take more time than expected to set up and configure a theme.
- It's easy to make mistakes like choosing the wrong colors, making text unreadable, etc.
- Users might add too many fonts to their themes, thereby spoiling the design and slowing down the website.
- More options means adding more functionality to the theme, thereby impacting on its performance.

Although your customers might not accept this at first, you are the theme designer, therefore you should be the one who is best placed to make the important design decisions for your theme.

A well-calibrated number of theme options should be sufficient to let your customers make some targeted and carefully predefined (by you) modifications to personalize the theme and make it their own.

Implement smart graphics optimization techniques

Images are likely to put the heaviest weight on theme performance compared to template files, scripts and styles. Just run any theme through Pingdom's Speed Test tool to verify this.

However, a generous use of full-screen, bold images is hardly surprising. Images add huge aesthetic value to themes and customers are drawn to a theme largely on the basis of its visual impact.

Although it's hard to resist the power of great graphics, the guidelines below can help you code faster themes without necessarily sacrificing aesthetics (too much):

- Using as few images as possible is always a good guiding principle.
- Take advantage of CSS blend modes, filters, and semi-transparent overlays on low-resolution images to mask pixelation and enhance their visual appearance. You can also try to experiment with CSS blend modes and filters on HTML elements and text rather than images to create stunning visual effects wherever possible.
- Make sure the images are of the appropriate format and optimized for web.
- Where appropriate, use CSS sprites to combine your images into a larger image and therefore reduce HTTP requests.
- Try implementing lazy loading for images. Doing so will achieve faster page load, and if users don't scroll to the location of the `<img>` tag, the image won't need to be downloaded at all. To lazy load images in WordPress you can choose from a number of good plugins, such as Lazy Load, a3 Lazy Load and Rocket Lazy Load.

Make customer support and education your pillars of strength

Instead of feeding your customers lots of confusing and time-consuming theme options, try offering full support and education on how to use your themes, image optimization, WordPress 101, etc.

You can do so using:

- blog posts
- downloadable guides
- video tutorials
- FAQs on your website
- a ticket system for more specific issues that might arise while using your themes.

Sharing the basic know-how about your products and the platform you build for is a great way of making your themes easy to use for your customers and contributes to generate a high level of trust, which is at the core of a healthy and successful business.

Sometimes Flexibility Wins over Performance

There may be a limited number of cases when you don't think compromising some flexibility for the sake of a small performance gain is a good idea.

A couple of examples come to mind.

When writing DRY code could make your customers unhappy

The first example comes from Tom McFarlin, a well-known WordPress theme developer and author.

In his blog he explains why he decided to include a long list of WordPress template files, thereby going against the DRY coding principle (and also

impacting negatively on performance to some extent) while developing his theme *Meyer*.

Meyer was targeted at bloggers and digital publishers without much WordPress technical knowledge. Knowing that lack of coding experience would not have stopped this audience from tinkering with his theme, Tom McFarlin thought of a way to make things easier for them. He provided his customers with a generous number of template files they could easily copy and paste into a WordPress child theme for customization without hacking the original theme.

McFarlin writes:

> If the templates are defined in the base theme — rather than, say, a bunch of developer-ish conditionals — users are going to have an easier time basing their templates off of something that already exists …
>
> So if I had to sum this up in some concise way, I'd say that I — like many developers — want to keep my code as DRY as possible, but there's a tradeoff that comes with doing so in the context of WordPress theme development.
>
> And I don't think that staying DRY is the best way to go about it if you're going to be marketing a product to a wide audience.
>
> Know Your Customer — the Blogger.

Static source code is better for performance but don't use it in your public themes

My second example of when not to compromise flexibility for some performance gain in your WordPress themes is based on performance advice from the Codex.

You'll find some great performance tips there, but there's one I wouldn't recommend you use, at least if you're going to release your theme to the general

public.

It goes like this:

- Can static values be hardcoded into your themes? This will mean you have to edit code every time you make changes, but for generally static areas, this can be a good trade off.
    - For example, your site charset, site title, and so on.
    - Can you hardcode menus that rarely change? Avoiding functions like `wp_list_pages()` for example.

Codex — WordPress Optimization/WordPress Performance

Avoiding unnecessary calls to the database and optimizing your SQL queries are excellent practices every theme should follow.

If your theme runs a bit slow, try looking for problematic database queries that take too long to execute. You can use a plugin like Query Monitor in your development environment to help you with this task.

However, hardcoding bits of information like site title, menu items, etc., is out of the question. Even if your users never change that info once it's set, they need to set it at least once and they rightly expect to do so using WordPress dynamic capabilities.

Furthermore, even if your theme's users were prepared to dig into the source code, they would have to hack the theme files to add the required data. This is almost never a good idea, not least because all changes would be overwritten on the next theme update.

Conclusion

Coding WordPress themes for the general public may involve catering for diverse and sometimes incompatible needs: those of your customer base, of web performance, hence the needs of visitors of those websites that use your themes, and even your own needs as WordPress theme designer and creator.

In this article I've expressed my views on how to address the dilemma WordPress theme developers might face between the demands of flexibility and those of performance.