Measure Performance with the RAIL Model

By Meggin Kearney

Meggin is a Tech Writer

By Addy Osmani

Eng Manager, Web Developer Relations

By Kayce Basques

Technical Writer for Chrome DevTools

By Jason Miller

Jason is a Web DevRel

RAIL is a **user-centric** performance model that breaks down the user's experience into key actions. RAIL's **goals** and **guidelines** aim to help developers and designers ensure a good user experience for each of these actions. By laying out a structure for thinking about performance, RAIL enables designers and developers to reliably target the work that has the highest impact on user experience.

Every web app has four distinct aspects to its life cycle, and performance fits into them in different ways:

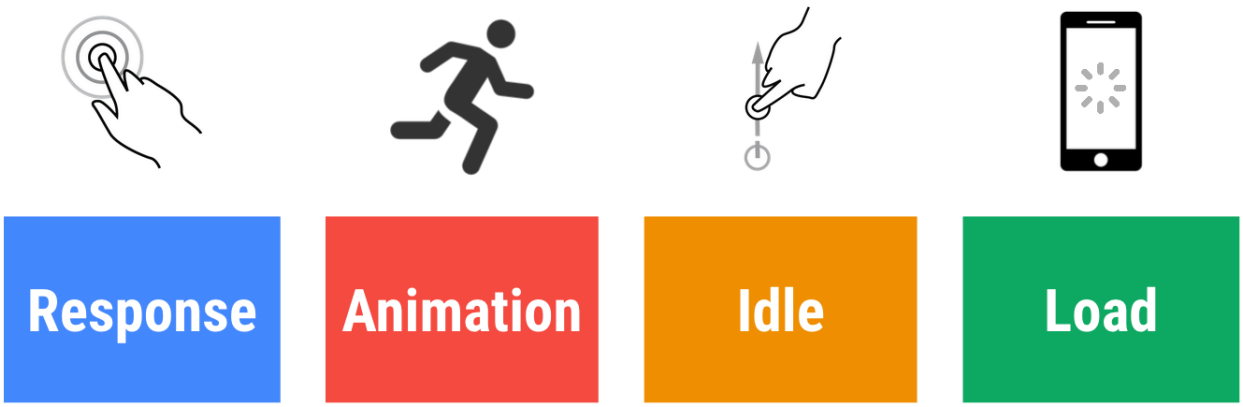

Figure 1. The 4 parts of the RAIL performance model

Goals and guidelines

In the context of RAIL, the terms **goals** and **guidelines** have specific meanings:

- **Goals**. Key performance metrics related to user experience. Since human perception is relatively constant, these goals are unlikely to change any time soon.

- **Guidelines**. Recommendations that help you achieve goals. These may be specific to current hardware and network connection conditions, and therefore may change over time.

Focus on the user

Make users the focal point of your performance effort. The table below describes key metrics of how users perceive performance delays:

User Perception Of Performance Delays

0 to 16ms	Users are exceptionally good at tracking motion, and they dislike it when animations aren't smooth. They perceive animations as smooth so long as 60 new frames are rendered every second. That's 16ms per frame, including the time it takes for the browser to paint the new frame to the screen, leaving an app about 10ms to produce a frame.
0 to 100ms	Respond to user actions within this time window and users feel like the result is immediate. Any longer, and the connection between action and reaction is broken.
100 to 300ms	Users experience a slight perceptible delay.
300 to 1000ms	Within this window, things feel part of a natural and continuous progression of tasks. For most users on the web, loading pages or changing views represents a task.
1000ms or more	Beyond 1000 milliseconds (1 second), users lose focus on the task they are performing.
10000ms or more	Beyond 10000 milliseconds (10 seconds), users are frustrated and are likely to abandon tasks. They may or may not come back later.

Users perceive performance delays differently, depending on network conditions and hardware. For example, loading an experience in 1000ms is plausible on a powerful desktop machine over a fast Wi-Fi connection, so users have grown accustomed to a 1000ms loading experience. But for mobile devices over slow 3G connections, loading in 5000ms is a more realistic goal, so mobile users are generally more patient.

Response: process events in under 50ms

Goal: Complete a transition initiated by user input within 100ms. Users spend the majority of their time waiting for sites to respond to their input, not waiting for the sites to load.

Guidelines:

- Process user input events within 50ms to ensure a visible response within 100ms, otherwise the connection between action and reaction is broken. This applies to most inputs, such as clicking buttons, toggling form controls, or starting animations. This does not apply to touch drags or scrolls.
- Though it may sound counterintuitive, it's not always the right call to respond to user input immediately. You can use this 100ms window to do other expensive work. But be careful not to block the user. If possible, do work in the background.
- For actions that take longer than 50ms to complete, always provide feedback.

50ms or 100ms?:

The goal is respond to input in under 100ms, so why is our budget only 50ms? This is because there is generally other work being done in addition to input handling, and that work takes up part of the time available for acceptible input response. If an application is performing work in the recommended 50ms chunks during idle time, that means input can be queued for up to 50ms if it occurs during one of those chunks of work. Accounting for this, it's safe to assume only the remaining 50ms is available for actual input handling. This effect is visualized in the diagram below which shows how input received during an idle task is queued, reducing the available processing time:

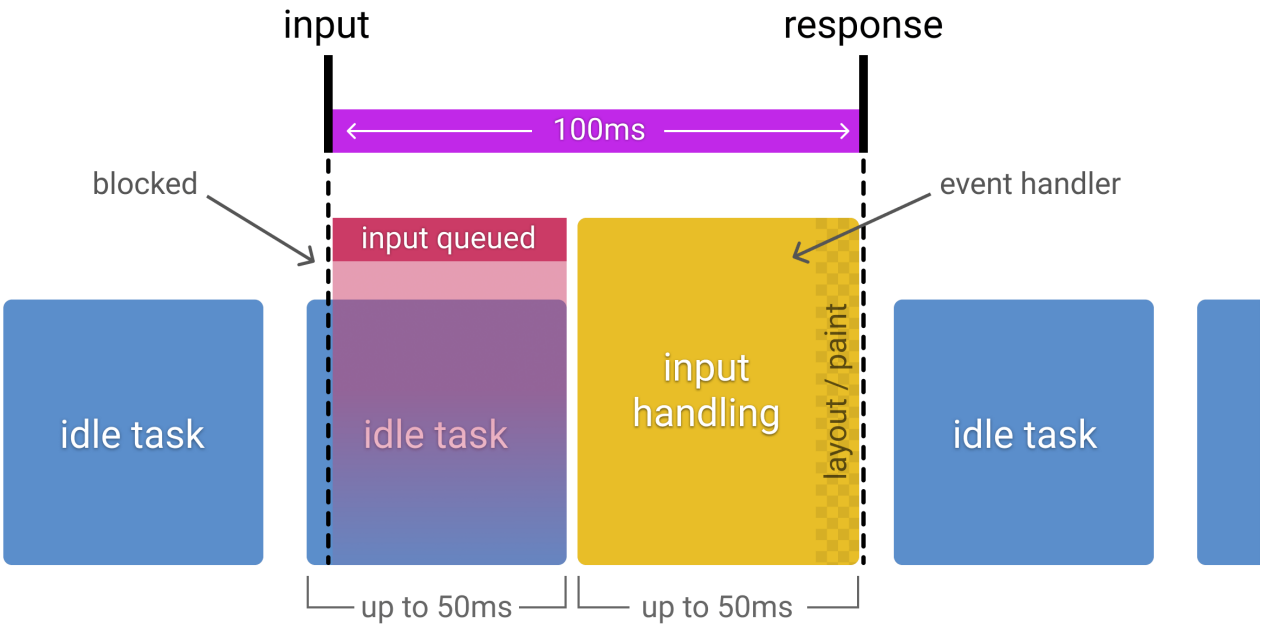

Figure 2. How idle tasks affect input response budget.

Animation: produce a frame in 10ms

Goals:

- Produce each frame in an animation in 10ms or less. Technically, the maximum budget for each frame is 16ms (1000ms / 60 frames per second ≈ 16ms), but browsers need about 6ms to render each frame, hence the guideline of 10ms per frame.
- Aim for visual smoothness. Users notice when frame rates vary.

Guidelines:

- In high pressure points like animations, the key is to do nothing where you can, and the absolute minimum where you can't. Whenever possible, make use of the 100ms response to pre-calculate expensive work so that you maximize your chances of hitting 60fps.
- See Rendering Performance for various animation optimization strategies.
- Recognize all the types of animations. Animations aren't just fancy UI effects. Each of these interactions are considered animations:
    - Visual animations, such as entrances and exits, tweens, and loading indicators.
    - Scrolling. This includes flinging, which is when the user starts scrolling, then lets go, and the page continues scrolling.
    - Dragging. Animations often follow user interactions, such as panning a map or pinching to zoom.

Idle: maximize idle time

Goal: Maximize idle time to increase the odds that the page responds to user input within 50ms.

Guidelines:

- Use idle time to complete deferred work. For example, for the initial page load, load as little data as possible, then use idle time to load the rest.
- Perform work during idle time in 50ms or less. Any longer, and you risk interfering with the app's ability to respond to user input within 50ms.
- If a user interacts with a page during idle time work, the user interaction should always take the highest priority and interrupt the idle time work.

Load: deliver content and become interactive in under 5 seconds

When pages load slowly, user attention wanders, and users perceive the task as broken. Sites that load quickly have longer average sessions, lower bounce rates, and higher ad viewability. See The Need For Mobile Speed: How Mobile Latency Impacts Publisher Revenue.

Goals:

- Optimize for fast loading performance relative to the device and network capabilities that your users use to access your site. Currently, a good target for first loads is to load the page and be interactive in 5 seconds or less on mid-range mobile devices with slow 3G connections. See Can You Afford It? Real-World Web Performance Budgets. But be aware that these targets may change over time.
- For subsequent loads, a good target is to load the page in under 2 seconds. But this target may also change over time.

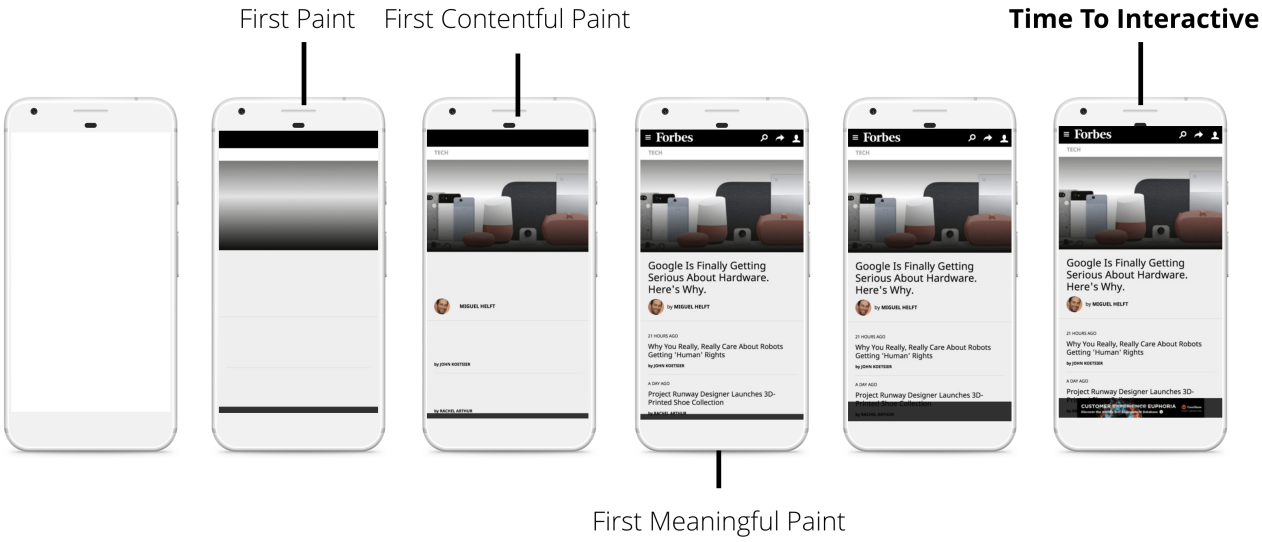

Figure 3. Each loading metric represents a different phase of the user's perception of the loading experience

Guidelines:

- Test your load performance on the mobile devices and network connections that are common among your users. If your business has information on what devices and network connections your users are on, then you can use that combination and set your own loading performance targets. Otherwise, The Mobile Economy 2017 suggests that a good global baseline is a mid-range Android phone, such as a Moto

G4, and a slow 3G network, defined as 400ms RTT and 400kbps transfer speed. This combination is available on WebPageTest.

- Keep in mind that although your typical mobile user's device might claim that it's on a 2G, 3G, or 4G connection, in reality the *effective connection speed* is often significantly slower, due to packet loss and network variance.
- Focus on optimizing the Critical Rendering Path to unblock rendering.
- You don't have to load everything in under 5 seconds to produce the perception of a complete load. Enable progressive rendering and do some work in the background. Defer non-essential loads to periods of idle time. See Website Performance Optimization.
- Recognize the factors that affect page load performance:
    - Network speed and latency
    - Hardware (slower CPUs, for example)
    - Cache eviction
    - Differences in L2/L3 caching
    - Parsing JavaScript

Tools for measuring RAIL

There are a few tools to help you automate your RAIL measurements. Which one you use depends on what type of information you need, and what type of workflow you prefer:

- **Chrome DevTools**. The developer tools built into Google Chrome. Provides in-depth analysis on everything that happens while your page loads or runs.
- **Lighthouse**. Available in Chrome DevTools, as a Chrome Extension, as a Node.js module, and within WebPageTest. You give it a URL, it simulates a mid-range device with a slow 3G connection, runs a series of audits on the page, and then gives you a report on load performance, as well as suggestions on how to improve. Also provides audits to improve accessibility, make the page easier to maintain, qualify as a Progressive Web App, and more.
- **WebPageTest**. Available at webpagetest.org/easy. You give it a URL, it loads the page on a real Moto G4 device with a slow 3G connection, and then gives you a detailed report on the page's load performance. You can also configure it to include a Lighthouse audit.

The sections below give you more information on how to use each tool to measure RAIL.

Chrome DevTools

The **Performance** panel is the main place to analyze your RAIL metrics. See Get Started With Analyzing Runtime Performance to get familiar with the **Performance** panel UI. The workflow and UI for analyzing load performance is mostly the same, the only difference is how you start and stop the recording. See Record load performance.

The following DevTools features are especially relevant:

- Throttle your CPU to simulate a less-powerful device.
- Throttle the network to simulate slower connections.
- View main thread activity to view every event that occurred on the main thread while you were recording.
- View main thread activities in a table to sort activities based on which ones took up the most time.
- Analyze frames per second (FPS) to measure whether your animations truly run smoothly.
- Monitor CPU usage, JS heap size, DOM nodes, layouts per second, and more in real-time with the **Performance Monitor**.
- Visualize network requests that occurred while you were recording with the **Network** section.
- Capture screenshots while recording to play back exactly how the page looked while the page loaded, or an animation fired, and so on.
- View interactions to quickly identify what happened on a page after a user interacted with it.
- Find scroll performance issues in real-time by highlighting the page whenever a potentially problematic listener fires.
- View paint events in real-time to identify costly paint events that may be harming the performance of your animations.

Lighthouse

See Get started to learn how to set up and run Lighthouse.

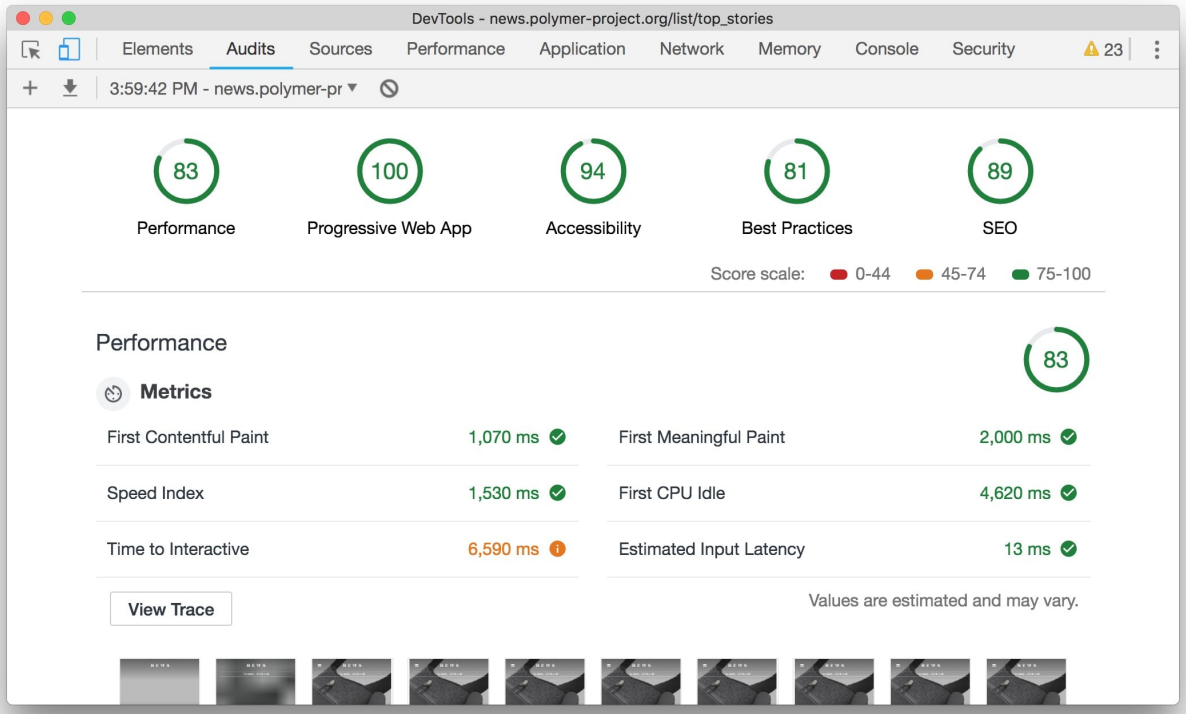

Figure 4. An example Lighthouse report

The following audits are especially relevant:

- **Response**
  - Estimated Input Latency. Estimates how long your app will take to respond to user input, based on main thread idle time.
  - Uses Passive Event Listeners To Improve Scrolling.
- **Load**
  - Registers A Service Worker. A service worker can cache common resources on a user's device, reducing time spent fetching resources over the network.
  - Page Load Is Fast Enough On 3G.
  - First Meaningful Paint. Measures when the page appears meaningfully complete.
  - First CPU Idle. Marks the first time at which the page's main thread is quiet enough to handle input.
  - Time To Interactive. Measures when a user can consistently interact with all page elements.
  - Perceptual Speed Index.
  - Reduce Render-Blocking Resources.

- Offscreen Images. Defer the loading of offscreen images until they're needed.
- Properly Size Images. Don't serve images that are significantly larger than the size that's rendered in the mobile viewport.
- Critical Request Chains. Visualize your Critical Rendering Path.
- Uses HTTP/2.
- Optimize Images.
- Enable Text Compression.
- Avoid Enormous Network Payloads.
- Uses An Excessive DOM Size. Reduce network bytes by only shipping DOM nodes that are needed for rendering the page.

WebPageTest

Enter a URL at webpagetest.org/easy to get a report on how that page loads on a real mid-range Android device with a slow 3G connection.

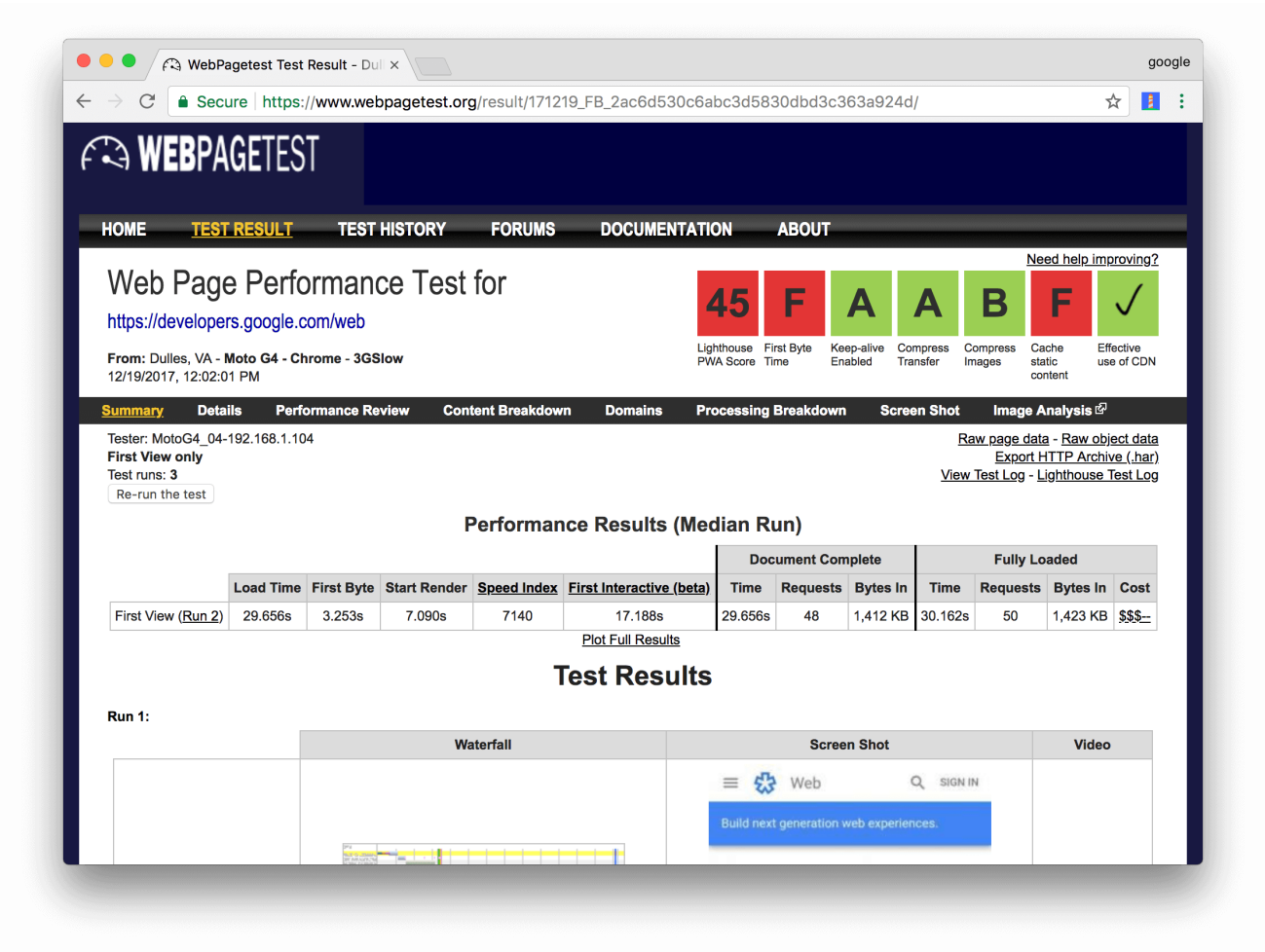

Figure 5. An example WebPageTest report

Summary

RAIL is a lens for looking at a website's user experience as a journey composed of distinct interactions. Understand how users perceive your site in order to set performance goals with the greatest impact on user experience.

- **Focus on the user**.
- **Respond to user input in under 100ms**.
- **Produce a frame in under 10ms when animating or scrolling**.
- **Maximize main thread idle time**.
- **Load interactive content in under 5000ms**.

Last updated July 19, 2018.