Building Resumable Applications with Qwik JS

Rik Robinson

@RikRobinson

where are we?

NEXT: Before we get into resumable apps and Qwik, let's first look at where we are now.
One quick metric to introduce so we have a common baseline for comparison.

Time to Interactive (TTI)

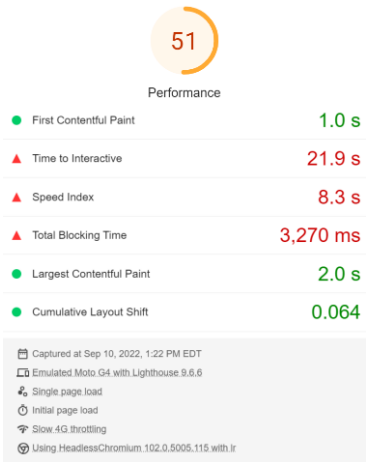
"The TTI metric measures the time from when the page starts loading to when its main sub-resources have loaded and it is capable of reliably responding to user input quickly."

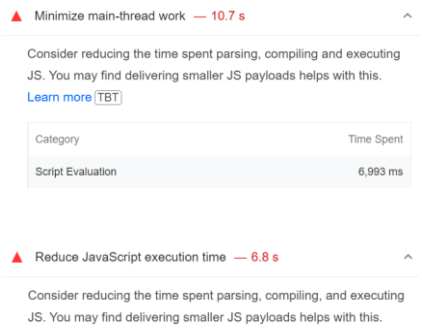
https://web.dev/tti/

SSR can lead to scenarios where a page *looks* interactive but it's not *actually* interactive because the main thread is blocked downloading the app re-hydrating it.

NEXT: Now that we have a name for what we are judging, let's take a look at top 3 e-commerce sites in the world and see where they are…

#3 - Alibaba.com

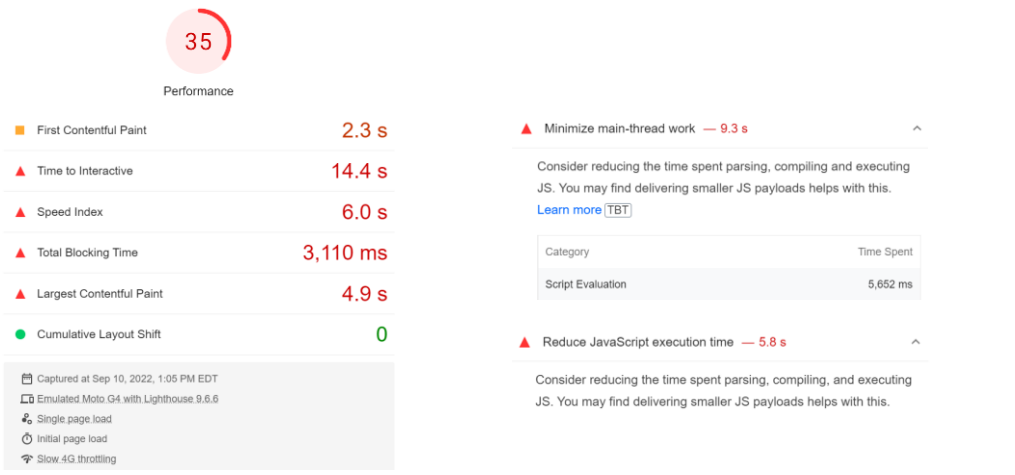




PageSpeed Insights

In 3rd place….

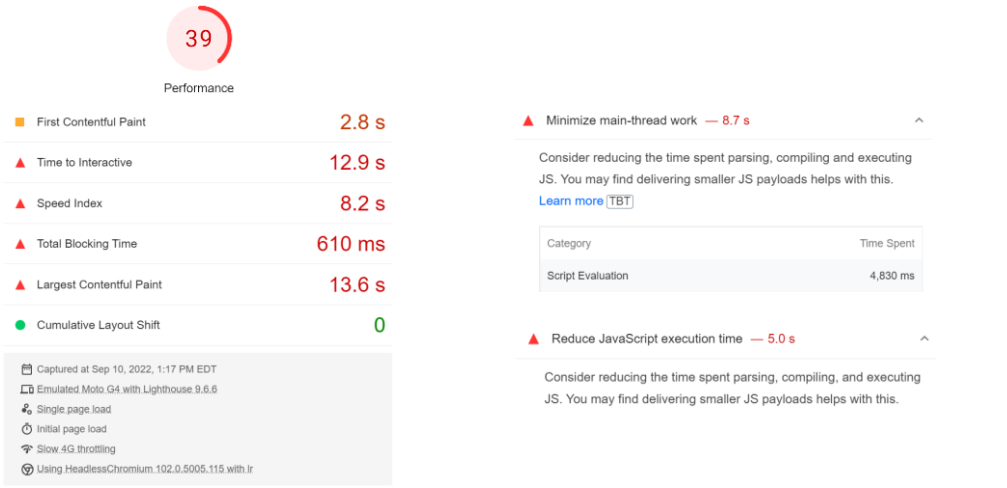
#2 - Ebay.com



PageSpeed Insights

In the #2 spot…

#1 - Amazon.com



#1 e-commerce site in the world

..ouch!

NEXT: PageSpeed offers optimization suggestions and, by far, the issue is JS

PageSpeed - Average Increase from Top 50 E-Commerce Sites

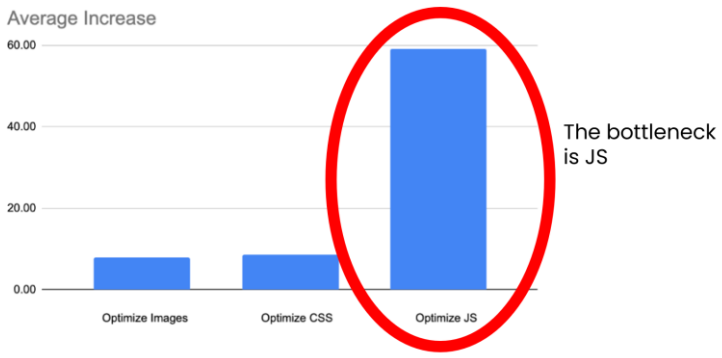


chart image: Miško Hevery (WebApps at Scale: ng-conf 2022) - https://t.co/LAHyP98RW7

PageSpeed offers several metrics for improvement, such as Optimize Images, Optimize CSS, and Optimize JS.
JS is consistently shown as the bottleneck across the top 50 e-commerce sites in a Builder.io study.
Pair this the fact that these top 50 have $$ to be the best, so imagine what the rest of e-commerce looks like.

- Note project I had for [large paper distrib] that started with Lighthouse of 3.
- Note that Amazon/ebay do not use any of the current frameworks

TTI !== EASY

So, obviously TTI is not easy

- These companies have big $$ and still losing the battle

https://unbounce.com/page-speed-report/

As page load time goes from:

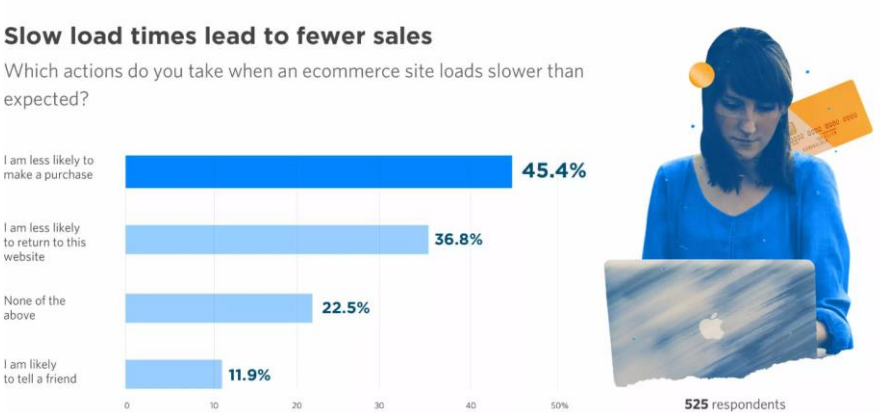
1s to 3s the probability of bounce **increases 32%**

1s to 5s the probability of bounce **increases 90%**

1s to 6s the probability of bounce **increases 106%**

1s to 10s the probability of bounce **increases 123%**

https://www.thinkwithgoogle.com/intl/en-gb/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/



https://unbounce.com/page-speed-report/

TTI ===

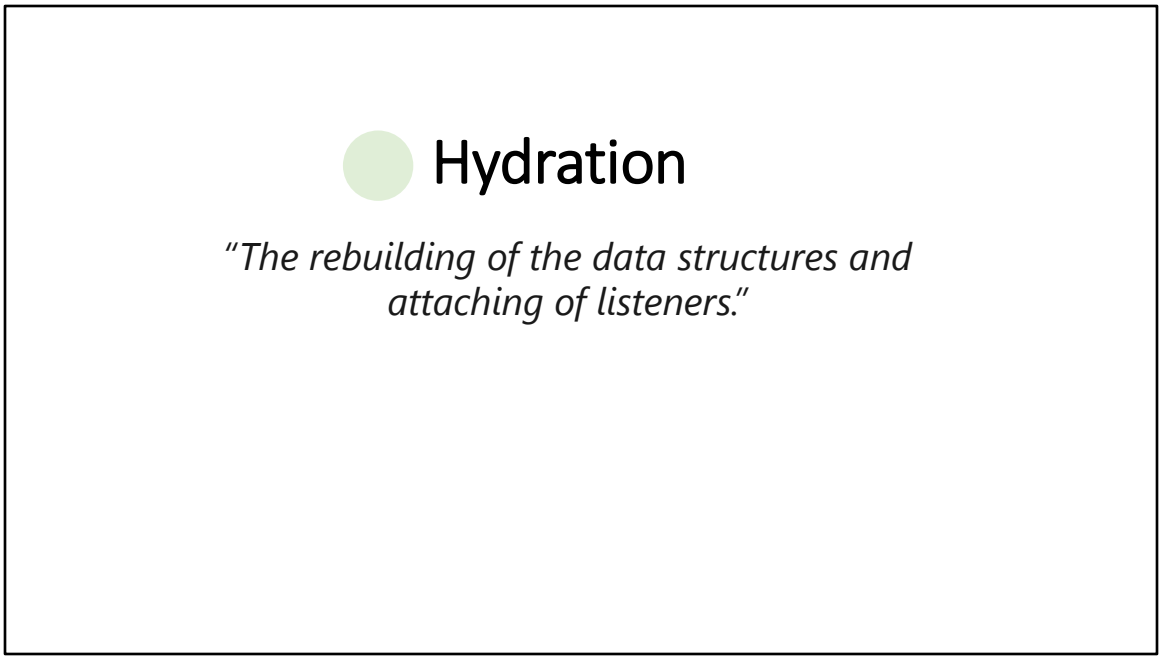
Improved TTI leads to more $$ ….

how did we get here?

History

- HTML first
- More interactivity (xmlHttpRequest/AJAX)
- Browser differences
- jQuery
- Component frameworks
- Third-party JS (analytics/marketing)
- Current (replayable) frameworks

- HTML First – JS sprinkled in for handlers (WAY WAY back: htc/js added behavior, xmlDataIsland for state).
- More interactivity – More application/desktop experience expected by users. xmlHttpRequest really changed the game.
- Browser differences – a lot of JS was added to detect browsers and code around their differences.
    - JS for enterprise became unmanageable and it was wild west out there with where JS should live and standards were hard to find
- jQuery - put an api around selection and removed some of the complexity of browser differences and dealing with xpath and such to select.
    - User experience continued to demand more and more interactions and animations and eventually spaghetti jQuery was born
    - no real component structure around the client code still.
- Component Frameworks
    - Backbone, KnockoutJS, AngularJS
    - Brings familiar server-side component paradigm to web JS
    - Focused on DX
    - Not focused on TTI
- Current Frameworks



•Hydration - What is it?

- Hydration is the rebuilding of the data structures and attaching of listeners.
- Application State - examples
- Framework State - examples

How do replayable frameworks work?

- Server runs application (SSR)
- Non-functional HTML version sent to browser
- JavaScript version of site sent to browser
- Browser parses JS version
- Browser reconciles HTML version with JS version
- Browser executes JS to create interactive version
- Browser replaces HTML version with interactive version
- User can interact

- Server runs application
- Generates the non-functional HTML version of the app and sends to browser
- Generates JS code to add listeners, reconstitute needed state, etc and sends to browser
- Browser parses JS
- Reconciliation step to verify the HTML version matches the JS version
- Run the JS to create the application and attach handlers, etc.
- Replace the HTML version in the browser before user interacts

Note that JS for ALL elements is sent regardless if ever used

- Rehydration is expensive

Replayable

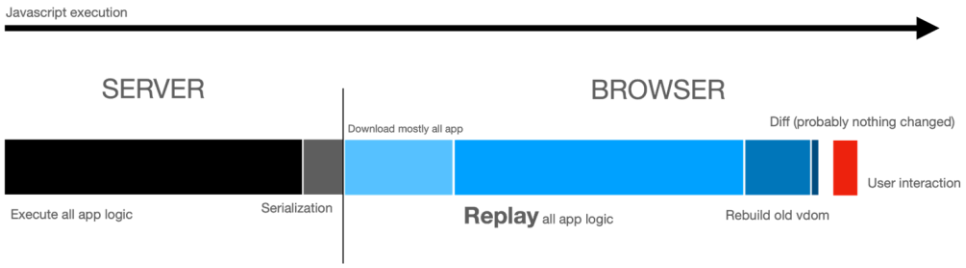


image: Miško Hevery (WebApps at Scale: ng-conf 2022) - https://t.co/LAHyP98RW7

This is the process of hydration

Current Frameworks

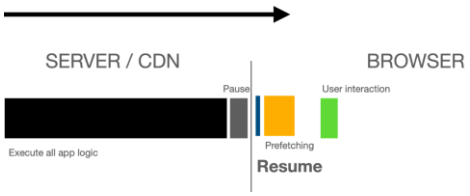
- PRO: Developer Experience (DX)
- PRO: Ecosystem of libraries/components
- PRO: Community of support
- PRO: Established patterns
- CON: Lazy-loading is hard
- CON: Framework not intimate with bundling
- CON: Initial bundle size increases with app size

Current (Replayable) Pros and Cons

- PRO: DX
- PRO: Ecosystem of libs/comps
- PRO: Community
- PRO: Established patterns
- CON: lazy-loading is hard because not built for it.
- CON: framework separate from bundling
- CON: init bundle size is O(n) and increases as app increases size/complexity.

NEXT: That is the number one con and not something easily overcome with replayable frameworks because of the way they are built from ground-up….

Resumable Applications

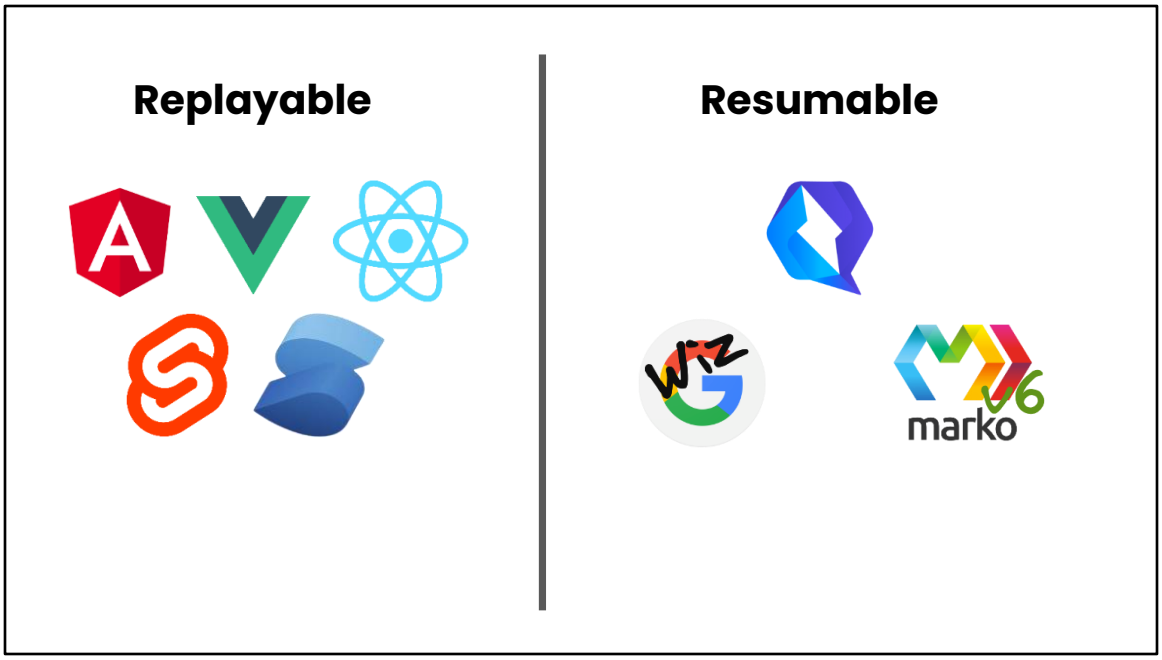


- Server runs application (SSR)
- Server serializes all state into HTML
- Server sends interactive version to browser
- User can interact

image: Miško Hevery (WebApps at Scale: ng-conf 2022) - https://t.co/LAHyP98RW7

- Server runs app
- Server serializes to HTML
- Server sends functional version to browser.

- Qwik serializes listeners, internal data structures, and framework state into the HTML.
- VM Analogy
    - Resumability allows Qwik applications to continue execution where the server left off. (skip the "boot" process)



Replayable:

- Angular, Vue, React, Solid, Svelte

Resumable:

* Qwik, G-Wiz, Marko

Qwik JS!

Qwik Overview

- General Purpose
- Resumable
- Bundling built-in
- Scalable
- Lazy Loading
- Reduced Rendering
- React-like Syntax
- SEO Friendly

- Qwik Overview
  - General Purpose – not just for e-commerce (example: EMS/Fire/Police dispatch initial load – slow could equal people dying)
  - Resumable – does not have to rehydrate, picks up where server left off on client
  - Bundling - Built from ground-up to address bundle size of delivered code
  - Scalable - Constant initial bundle size regardless of size of app
  - Lazy loading – JS, HTML, styles, configurable pre-fetching
  - Reduced Rendering – surgical precision on what is rendered and when
  - React Like syntax – very similar to functional React
  - SEO Friendly – site is ready faster – Google signals (load time, mobile friendly, secure)

Qwik Goals

- Only download and execute the bare minimum of the application.
- Delay execution and download of JS as long as possible.
- Serialize the execution state of the application and the framework on the server and resume it on the client.

•Think Qwik

- The goal of Qwik is having only to download and execute the bare minimum of the application.
- Delay execution and download of JavaScript for as long as possible.
- Serialize the execution state of the application and the framework on the server and resume it on the client.

Demo

Qwik JS

Qwik Advanced Concepts

- Prefetching
- Containers
- Qwikify for React components (later)
- SSR Streaming
- Built-in Debugging (SSR, browser, prerender)
- Built-in Extendable Styling (Tailwind, PostCss, …)
- Ready for Edge Caching (Netlify, Cloudflare, …)

- Advanced
  - Prefetching – configurable strategies, not the same as resumable (does not execute)
  - Containers – to break app into smaller parts, can comm and share data with others
  - Qwikify for React components coming later
  - SSR Streaming
  - Debugging (SSR, browser, prerender)
  - Extendable styling (Tailwind, PostCss, …)
  - Edge caching (Netlify, Cloudflare)

Qwik City

- Companion meta-framework for Qwik
- Directory-based routing
- Nested layouts
- Breadcrumbs
- MDX authoring
- File-based menu definition
- Data endpoints
- Currently MPA only - SPA next

Not demoing, just mentioning. Compare to Remix/Next for React.

- Companion meta-framework for Qwik
- Directory-based routing
- Nested layouts
- Breadcrumbs
- MDX authoring
- File-based menu definition
- Data endpoints
- Current MPA only – SPA next

Partytown

Partytown is a lazy-loaded library to help relocate resource intensive scripts into a web worker, and off of the main thread.

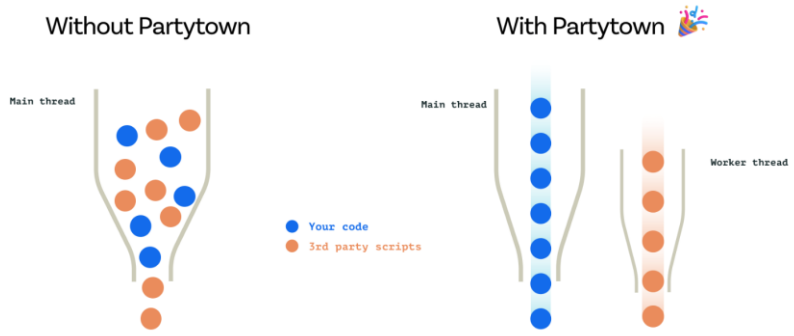


image from partytown.builder.io

Not demoing, just mentioning

- Quick aside on how one might handle 3rd-party JS
- Separate from Qwik
- Partytown moves 3rd party scripts to webworker and off main threads
- How they do it is very interesting - encourage you to take a look at PT – resources have a link to how it works post

NEXT: Now, back to our scripts and code…

Get Started with Qwik

- Download at qwik.builder.io
- Docs, Examples, Playground, Tutorials
- Discord community (qwik.builder.io/chat)
- Twitter (@QwikDev)
- BETA will be soon

- Download
- Docs, Examples, Playground, Tutorials
- Discord – very helpful smart people and the team/creators are there and listening
- Twitter
- BETA soon

Resources

- Qwik Docs
  - https://qwik.builder.io/docs/overview/
- Hydration is Pure Overhead
  - https://www.builder.io/blog/hydration-is-pure-overhead
- Don’t Blame the Developer for What the Frameworks Did
  - https://www.builder.io/blog/dont-blame-the-developer-for-what-the-frameworks-did
- Our Current Frameworks are O(n); We Need O(1)
  - https://www.builder.io/blog/our-current-frameworks-are-on-we-need-o1
- How Partytown Works
  - https://www.builder.io/blog/how-partytown-works
- Deliotte: Milliseconds Make Millions
  - https://www2.deloitte.com/content/dam/Deloitte/ie/Documents/Consulting/Milliseconds_Make_Millions_report.pdf

Recommended Presentations

- JS-Poland: Miško with additional demos (2022-09-07)
  - https://youtu.be/7MgNMIPISY4
- Learn w/ Jason: Miško walks Jason through new Qwik app (2022-05-01)
  - https://youtu.be/_PDpoJUacuc
- Pure HTML Streaming with Qwik (2022-08-11)
  - https://youtu.be/yVOI81GKZBo

questions?

Rik Robinson

@RikRobinson