HTMX

create reactive web applications
with (almost) no JavaScript

Frontend history

Asbjørn Ulsberg
@bitbear@icosahedron.website

The untold history of web development:

1990: HTML invented.
1994: CSS invented to fix HTML.
1995: JS invented to fix HTML/CSS.
2006: jQuery invented to fix JS.
2010: AngularJS invented to fix jQuery.
2013: React invented to fix AngularJS.
2014: Vue invented to fix React & Angular.
2016: Angular 2 invented to fix AngularJS & React.
2019: Svelte 3 invented to fix React, Angular, Vue.
2019: React hooks invented to fix React.
2020: Vue 3 invented to fix React hooks.
2020: Solid invented to fix React, Angular, Svelte, Vue.
2020: HTMX 1.0 invented to fix React, Angular, Svelte, Vue, Solid.
2021: React suspense invented to fix React, again.
2023: Svelte Runes invented to fix Svelte.
2024: jQuery still used on 75% of websites.

(By twitter.com/fireship_dev)

The web is getting slower

LCP reports the render time of the largest image, text block, or video visible in the viewport.

LCP

Largest Contentful Paint

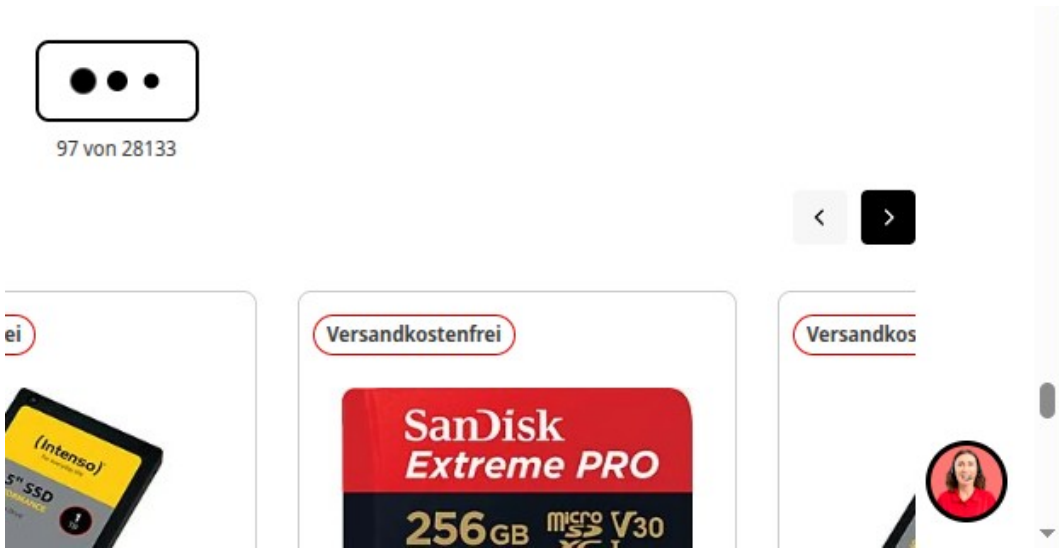
FCP measures the time from when the user first navigated to the page to when any part of the page's content is rendered.

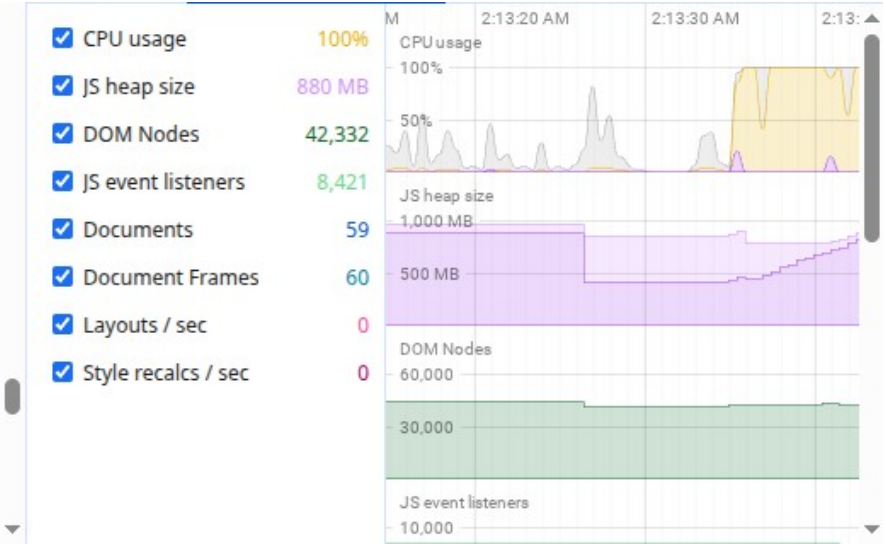
FCP

First Contentful Paint

The web is getting too slow

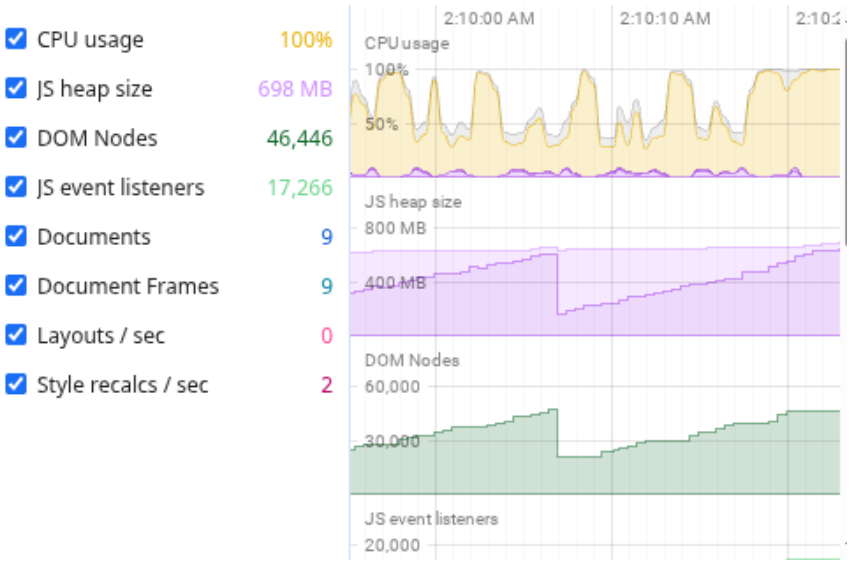
typical e-commerce website



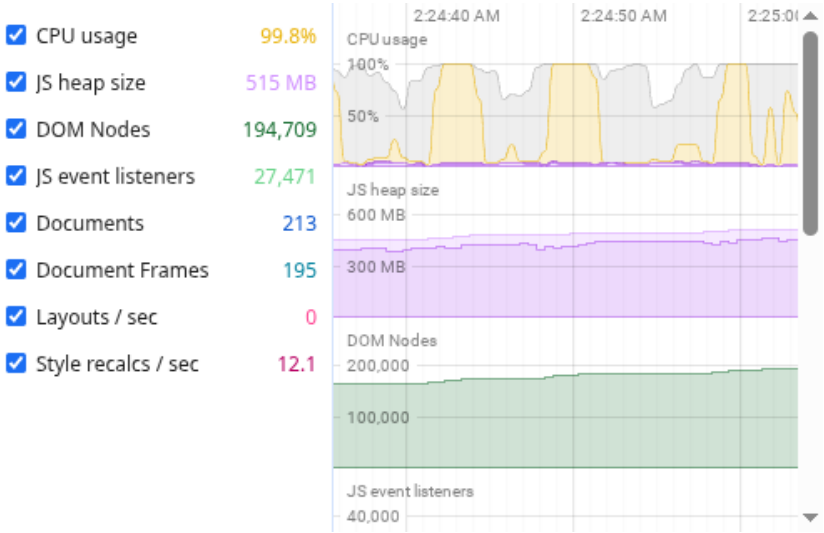


The web is getting too slow

facebook.com



linkedin.com



What is reactivity?

A mechanism

- that allows a web application's user interface
- to automatically update and react in response to changes in its underlying data or state without manual intervention
- that ensures the UI is always synchronized with the data
- that provides a dynamic and seamless user experience

How does reactivity work?

source

How Reactivity Works in Vue

We can't really track the reading and writing of local variables like in the example. There's just no mechanism for doing that in vanilla JavaScript. What we **can** do though, is intercept the reading and writing of **object properties**.

There are two ways of intercepting property access in JavaScript: getter / setters and Proxies. Vue 2 used getter / setters exclusively due to browser support limitations. In Vue 3, Proxies are used for reactive objects and getter / setters are used for refs. Here's some

source

Runtime vs. Compile-time Reactivity

Vue's reactivity system is primarily runtime-based: the tracking and triggering are all performed while the code is running directly in the browser. The pros of runtime reactivity are that it can work without a build step, and there are fewer edge cases. On

Limitations of reactivity

Limitations of `reactive()`

The `reactive()` API has a few limitations:

1. **Limited value types:** it only works for object types (objects, arrays, and collection types such as `Map` and `Set`). It cannot hold primitive types such as `string`, `number` or `boolean`.

2. **Cannot replace entire object:** since Vue's reactivity tracking works over property access, we must always keep the same reference to the reactive object. This means we can't easily "replace" a reactive object because the reactivity connection to the first reference is lost:

3. **Not destructure-friendly:** when we destructure a reactive object's primitive type property into local variables, or when we pass that property into a function, we will lose the reactivity connection:

source

vue.js and SSR – double hydration

To make the client-side app interactive, Vue needs to perform the **hydration** step. During hydration, it creates the same Vue application that was run on the server, matches each component to the DOM nodes it should control, and attaches DOM event listeners.

Hydration Mismatch

If the DOM structure of the pre-rendered HTML does not match the expected output of the client-side app, there will be a hydration mismatch error. Hydration mismatch is most commonly introduced by the following causes:

The data used during render contains randomly generated values. Since the same application will run twice - once on the server, and once on the client - the random values are not guaranteed to be the same between the two runs. There are two ways

The server and the client are in different time zones. Sometimes, we may want to convert a timestamp into the user's local time. However, the timezone during the

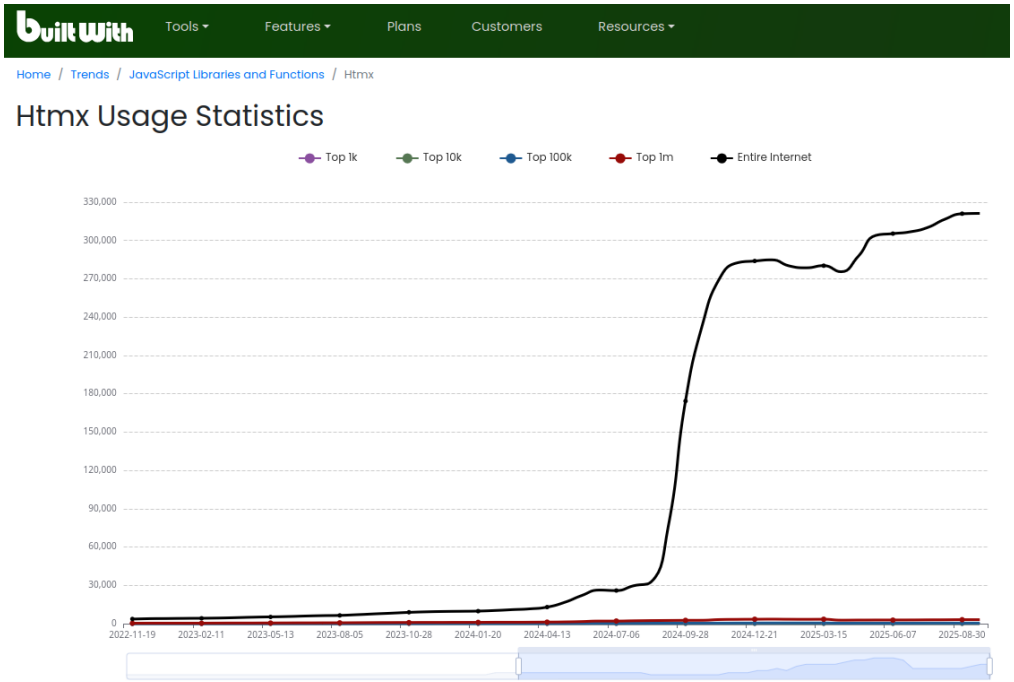
source

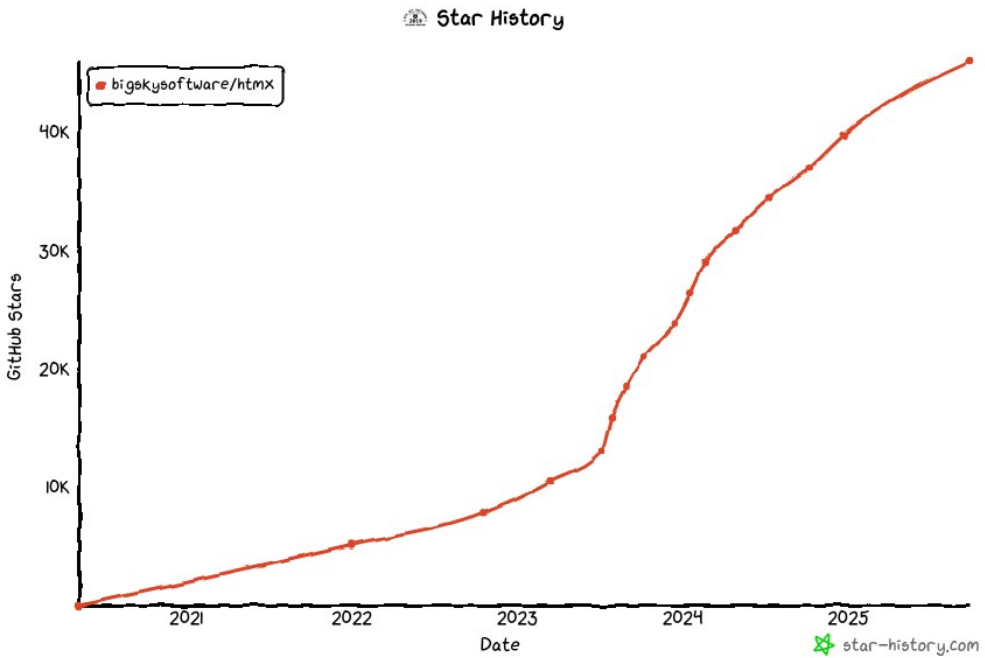
What is HTMX?

- a small JavaScript **library** (16 kb compressed)
- **extends HTML** with a few **attributes**
- allows you to **update parts of a web page**
- with HTML response from the server
- enables AJAX requests on all tags with all HTTP request methods
- provides reactive experience without reactivity

➤ open source, 46k stars on github

Statistics


built with
Tools
Features
Plans
Customers
Resources
Home / Trends / JavaScript Libraries and Functions / Htmx
Htmx Usage Statistics
Top 1k
Top 10k
Top 100k
Top 1m
Entire Internet
330,000
300,000
270,000
240,000
210,000
180,000
150,000
120,000
90,000
60,000
30,000
0
2022-11-19
2023-02-11
2023-05-13
2023-08-05
2023-10-28
2024-01-20
2024-04-13
2024-07-06
2024-09-28
2024-12-21
2025-03-15
2025-06-07
2025-08-30


Star History
bigskysoftware/htmx
40K
30K
20K
10K
GitHub Stars
2021
2022
2023
2024
2025
Date
star-history.com

Example

```
<script src="https://cdn.jsdelivr.net/npm/htmx.org@2.0.7/dist/htmx.min.js"></script>

<a hx-get="/backend-giving-html-response" hx-target="#fooId"> Click me </a>

<form hx-post="/backend-giving-html-response" hx-target="#fooId">
    <div>Foo: <input type="text" name="foo"></div>
    <div><input type="submit" value="Click me"></div>
</form>

<button hx-delete="/backend-giving-html-response" hx-target="#fooId"> Click Me </button>

<div id="fooId"></div>
```

Why HTMX?

- HTMX overcomes HTML limitations
  - supports PUT, PATCH, and DELETE methods in forms, buttons, links
  - adds target attribute to place the HTML response in any DOM element
  - allows to **update parts** of a web page
- **no compiling**, no bundling, no frontend tool chain
- reactive application experience
  - with server side HTML rendering
  - without writing client side JS
  - without APIs, schemas, etc.

Why HTMX?

- focus
  - **on frontend design** with CSS (styling, animations, etc.)
  - **on business logic** and input validaiton in the backend
- only transfer HTML from backend
- extreme **fast loading and rendering** (pure HTML is always fast)
- no client state, no hydration, no dehydration of data
- **no memory leaks** in the frontend
- **no JS errors** when your data is not correct

Why HTMX?

- no apis, no api documentation, no schemas, no json serializing
- no client side models, routers, controllers, stores, sessions, views
- no version mismatches between frontend and backend
- error messages, error handling are all server side
- easy to monitor, clear HTTP logs, all business logic server side

➤ **less complexity**

➤ **quicker to develop**, **easier to test, easier to maintain**

➤ **freedom to choose** any backend technology

HTMX configuration

```
<meta name="htmx-config" content='{
    "timeout":30000,
    "allowScriptTags":false,
    "refreshOnHistoryMiss": true,
    "responseHandling": [{"code":".*", "swap": true}]
}'>
```

```
<body hx-headers="js:{'X-CSRF-TOKEN': '{{ $request->session()->token() }}'}"
      hx-history="false">
```

```
<script>
    window.addEventListener('htmx:sendError', (e) => alert(e.detail.error + ' ' + e.detail.xhr.responseURL));
    window.addEventListener('htmx:timeout', (e) => alert(e.detail.error + ' ' + e.detail.xhr.responseURL));
</script>
```

JS plugin usage

Initialize

```
<script>
    window.addEventListener('htmx:sendError', (e) => alert(e.detail.error + ' ' + e.detail.xhr.responseURL));
    window.addEventListener('htmx:timeout', (e) => alert(e.detail.error + ' ' + e.detail.xhr.responseURL));
    document.addEventListener('htmx:load', (e) => {
        e.target.querySelectorAll('select.tomselect').forEach(el => initTomSelect(el));
        e.target.querySelectorAll('div.photoswipe').forEach(el => initPhotoSwipe(el));
        e.target.querySelectorAll('canvas.chartjs').forEach(el => initChart(el));
    });
</script>
```

Destroy

```
function initTomSelect(element) {
    const tomselect = new TomSelect(element, { ... });

    htmx.on(element, 'htmx:before-cleanup-element', event => tomselect.destroy(), {once: true});
}
```

Server side HTML rendering

- Template engines (Blade, Twig, etc.)
- Generate HTML strings manually
- Spatie Laravel-HTML (PHP/Laravel library)

```php
$form = html()->form('PUT', '/endpoint')
              ->addChildren([
                  html()->text('some-name'),
                  html()->number('some-number'),
                  html()->submit('Save')
              ])
              ->attributes(['hx-post' => '/some-url', 'hx-target' => 'some-dom-target'])
              ->class('some-css-class');

return $form;
```

Advanced features

Submit form on input change and submit

```
'hx-trigger' => 'submit queue:none, change queue:last',
```

`queue:<queue option>` - determines how events are queued if an event occurs while a request for another event is in flight. Options are:

- `first` - queue the first event
- `last` - queue the last event (default)
- `all` - queue all events (issue a request for each event)
- `none` - do not queue new events

➤ can block or queue concurrent requests

Advanced features #2

Loading indicators

```
.htmx-request {
    &:is(button) {
        background-color: initial !important;
        color: #ccc !important;
    }
    &:is(a) {
        color: #ccc !important;
    }
}
```

Advanced features #3

Conditional triggers

```
'hx-trigger' => 'click[!document.getSelection().toString() && !ctrlKey]',
```

Full page rendering vs. partial content rendering

```
@if ($request->headers->has('HX-Request'))
    // render partial content
@else
    // render full page
@endif
```

Advanced features #4

Custom events

```
<div hx-get="/get-fresh-data" hx-trigger="MyRefreshTable" hx-target="this">

    <a hx-delete="/delete-some-data" hx-target="this"> Delete </a>

    <!-- SOME TABLE DATA -->
</div>
```

backend:

```
// DELETE data

return response('Deleted!', 200, headers: ['HX-Trigger' => ['MyRefreshTable']]);
```

Advanced features #5

Multiple targets from one request: out-of-band

```
<div hx-trigger="MyUpdateEvent" hx-get="/url" hx-swap="none" hx-select-oob="#product-error, #product-data">
    <div id="product-error">
        ...
    </div>

    <div id="product-data">
        ...
    </div>
</div>
```

Multiple targets from one response:

```
Frontend:
<div class="notifications"></div>

...

Response:
<div hx-swap-oob="beforeend:.notifications">  <div>Operation success!</div>  </div>
```

Advanced features #6

Content swapping strategies: hx-swap

- innerHTML (default)
- outerHTML
- textContent
- none
- beforeend: insert after last child of target element
- afterend: insert after target element

Security

- HTML user input can contain hx-attributes
- Sandbox user input given as HTML:

```
<iframe srcdoc="{{ $userInputWithHTML }}" sandbox></iframe>
```

Limitations

- no offline mode built-in
- backend required
- development more backend-centric
- canvas rendering, custom form elements, etc. still require JS plugins
- can’t use vue.js, React UI component libraries
- requires reading the documentation carefully

Summary

HTMX

- enables reactive pages without React, Vue, etc.
- with (almost) no JS code
- doesn’t require compiling, bundling
- doesn’t require APIs, schemas
- helps to build very fast, stable and scalable frontends

➤ can significantly improve your productivity

➤ makes backend developers very happy!

Resources

- HTMX for Impatient Devs
- HTMX in 100 seconds
- The story of HTMX
- HTMX documentation
- Why you should choose HTMX for your next project
- A Real World React -> htmx Port
- Laravel HTML library
- IFrame sandbox attribute
- A Look At HTMX With PHP

Thank You!

Questions?