=> htmz

a low power tool for html

htmz is a minimalist HTML microframework that gives you the power to create modular web user interfaces with the familiar simplicity of **plain HTML**. [GitHub]

plain

Use straight up HTML. No supersets. No hz- ng- hx- v- w- x-; no special attributes. No DSLs. No <custom-elements>. *Just vanilla HTML.*

lightweight

176 bytes in total. Zero dependencies. Zero JS bundles to load. Not even a backend is required. *Just an inline HTML snippet*.

nofilter

No preventDefaults. No hidden layers. Real DOM, real interactions. No VDOM, no click listeners. No AJAX, no fetch. *No reinventing browsers*.

In a nutshell, htmz lets you swap page fragments using vanilla HTML code.

Imagine clicking a link, but instead of reloading the whole page, it *only updates the relevant portion of the page*.

htmz is an *experiment* inspired by htmx, Comet, ‘HTML As The Engine Of Application State’[1][2], and other similar web application architectures.

Demos

Check out these demos to get an idea of what htmz can do!

Tabs Greeting Edit inline Dialog More examples#

🐙 Select an example above!

Installing

Simply copy the following snippet into your page:

```
<iframe hidden name=htmz onload="setTimeout(()=>document.query
Selector(this.contentWindow.location.hash||null)?.replaceWith
(...this.contentDocument.body.childNodes))"></iframe>
```

For **npm enjoyers**, the following npm commands automate the process of copying the snippet into your page:

```
npm install --save-dev htmz
npx htmzify ./path/to/my/index.html
```

For **hackers**, you may start with the development version (deminified): htmz.dev.html

Basic usage

To invoke htmz, you need a hyperlink (or form) having these attributes:

1. href (or action) pointing to the **resource URL** `href="/flower.html⋯`
2. Continuing within the href: **destination ID selector** `⋯#my-element"`
3. And a **target** attribute with this value `target=htmz`

```
<!-- Loads /flower.html onto #my-element -->
<a href="/flower.html#my-element" target=htmz>Flower</a>
```

While this looks like an abuse of the URL fragment (it is), there is no other use for the URL fragment in this context, so it was repurposed as the destination ID selector. And it already looks like a CSS ID selector.

⚠ **Important note:** The loaded content **replaces** the selected destination. It may not be intuitive at first, but htmz does *not* insert the content into the destination. The rationale is that replacement is a more powerful operation. With replacement, you can *replace*, *delete* (replace with nothing), and *insert-into* (replace with the same container as original).

What does it do exactly?

htmz does one thing and one thing only.

> **Enable you to load HTML resources within *any element* in the page.**

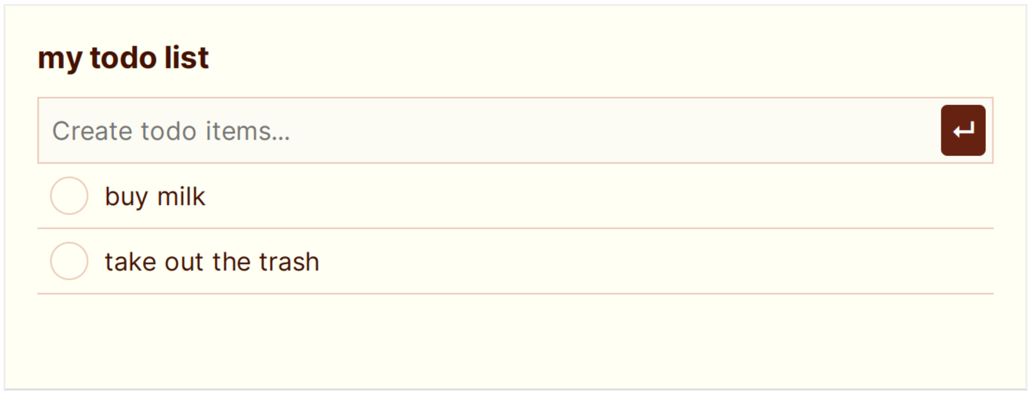
Think tabbed UIs, dual-pane list-detail layouts, dialogs, in-place editors, and the like.

This idea is not new. Dividing web pages into independently reloading parts has been a thing since mid-1990s. They were called **frames**, namely, <iframe>s, <frame>s, and <frameset>s.

htmz is a generalisation of HTML frames. — Load HTML resources within ~~any frame~~ any element in the page.

Read more on how it works in a section below.

Examples

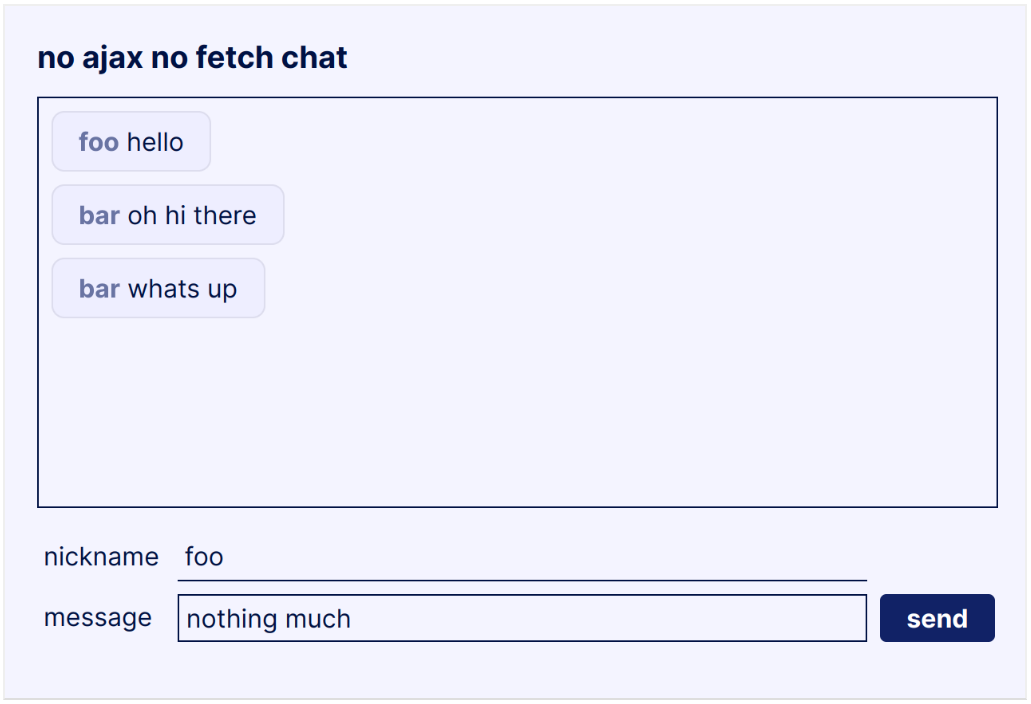


my todo list

Create todo items...

buy milk

take out the trash



no ajax no fetch chat

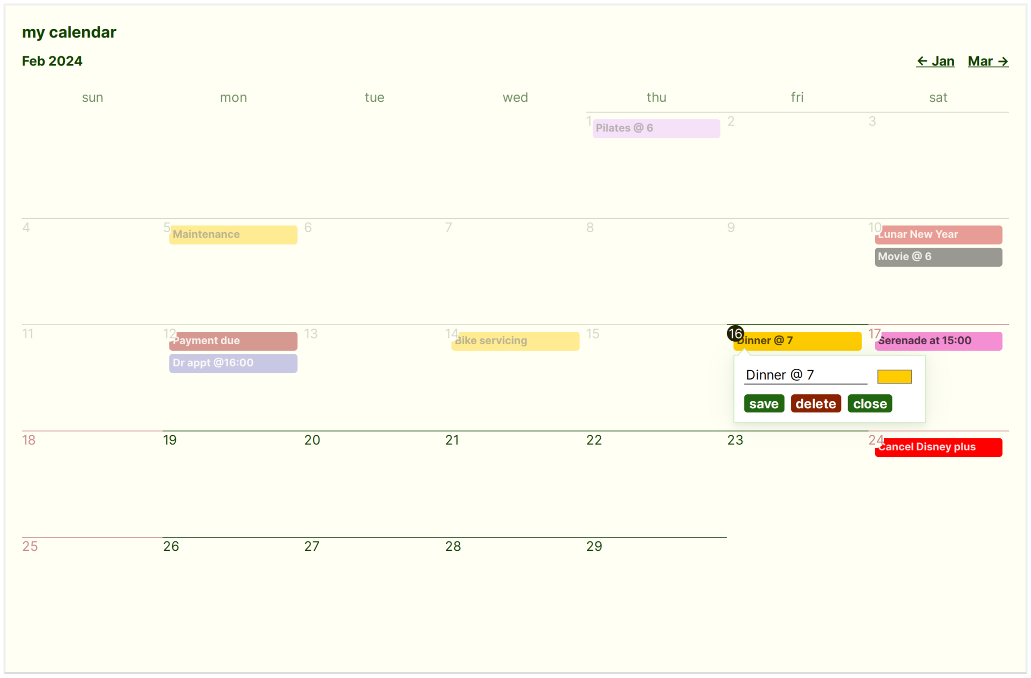
foo hello

bar oh hi there

bar whats up

nickname foo

message nothing much



my calendar

Feb 2024

sun	mon	tue	wed	thu	fri	sat
				1 Pilates @ 6	2	3
4	5 Maintenance	6	7	8	9	10 Lunar New Year Movie @ 6
11	12 Payment due Dr appt @16:00	13	14 Bike servicing	15	16 Dinner @ 7	17 Serenade at 15:00
18	19	20	21	22	23	24 Cancel Disney plus
25	26	27	28	29		

Dinner @ 7

More example applications, componentization approaches, and code in different languages can be found in the `/examples` directory. To start the example server:

```
cd examples
./run_servers.sh
```

Then load `http://localhost:3000/`.

Advanced usage

Naturally, only <a> and `<form>` elements can target and invoke htmz (as of current HTML5). This is fine; it's semantic, after all. However, HTML offers a couple more features that work well with htmz.

Per-button action & target

If you want to override the form's action on a per-button basis, use the `<button>`'s `formaction` attribute.

```
<form action="/default#my-target" target=htmz>
  <button>Default form action</button>
  <button formaction="/button#my-target">
    Different button action
  </button>
  <button formaction="/another-action#another-target">
    Another action
  </button>
</form>
```

Base target value

Tired of adding `target=htmz` to every link and form?

Using the base element, set htmz as the default target for all relative links. Add this at the top of your page.

```
<base target=htmz>
```

Clean target values

Don't like the look of `target=htmz` at all? Prefer using the real target as the value?

We can do a hack that enables you to write the target ID selector in the target attribute itself! Like this:

```
<!-- Loads /flower.html onto #my-element -->
<a href="/flower.html" target="#my-element">Flower</a>
```

The key is to add an iframe with a *matching name*, and modify the htmz snippet accordingly.

```
<iframe hidden name="#my-element" onload="htmz(this)"></iframe>
<script>
  function htmz(frame) {
    document.querySelector(frame.name) // use the iframe's name instead of the
      ?.replaceWith(...frame.contentDocument.body.children);
  }
</script>
```

You can even automate the generation of matching target iframes.

Scripting / interactivity

If you need something more interactive than the request-response model, you may try the htmz companion scripting language: **javazcript**. Sorry, I meant JavaScript, a scripting language designed to make HTML interactive.

htmz does not preclude you writing JS or using UI libraries to enhance interaction. You could, say, enhance a single form control with vanillaJS, but the form *values* could still be submitted as a regular HTTP form with htmz.

That said, htmz is extensible!

Extensibility

Need advanced selectors? Need error handling? Multiple targets? Fear not; the hero is here to save the day. The hero is you.

Here’s the development version of the snippet. Feel free to hack and extend according to your needs. You’re a programmer, right?

```
<script>
  function htmz(frame) {
    // Write your extensions here

    // Remove setTimeout to let the browser autoscroll content changes into vi
    setTimeout(() =>
      document
        .querySelector(frame.contentWindow.location.hash || null)
        ?.replaceWith(...frame.contentDocument.body.children)
    );
  }
</script>
<iframe hidden name=htmz onload="htmz(this)"></iframe>
```

A number of extensions will be available in the custom builder (coming soon!).

FAQ

How does it work?

htmz is an iframe named "htmz". You invoke htmz by loading a URL into the iframe via target=htmz. By using an iframe, we lean on the browser's native capability to fetch the URL and parse the HTML. After loading the HTML resource, we take the resulting DOM via an onload handler.

htmz is essentially a **proxy target**.

Like how a proxy server forwards requests to some specified server, proxy target htmz forwards responses into some specified target.

```
<!-- The ideal:
     GET /flower.html => #my-element            -->
<a href="/flower.html" target="#my-element">Flower</a>

<!-- Actual:
     GET /flower.html =htmz> #my-element        -->
<a href="/flower.html#my-element" target=htmz>Flower</a>
```

When you load a URL into the htmz iframe, the onload handler kicks in. It extracts your destination ID selector from the URL hash fragment and transplants the iframe’s contents (now containing the loaded HTML resource) into your specified destination.

htmz only runs when you invoke it. It does not continually parse your DOM and scan it for special attributes or syntax, nor does it attach listeners in your DOM. It’s a proxy not a VPN.

So it’s just another JavaScript framework?

Oh my! Not the f-word!!!

On a more serious note, I would say that rather than a JS one, it’s more of an HTML micro-f*******k. It does use JS, but only the minimum necessary.

Is htmz a library or a framework?

htmz is a snippet. ✂

What does htmz mean?

HTMZ stands for ***H****tml with* ***T****argeted* ***M****anipulation* ***Z****ones.*

Is this a joke?

This started as a *“Do I really need htmx? Can’t I do the load-link-into-target thing with current web? Sounds a lot like frames.”* and ended up with this.

So, it isn’t quite a joke, but a response to htmx. I wanted to try htmx. The premise sounded great (*Why should you only be able to replace the entire screen?*), then I saw that it was 16kB of JavaScript. Huh. Then there’s special syntax everywhere. Huh. I don’t want to learn a whole new set of instructions and Turing-complete DSLs specific to those instructions.

Regardless of joke status, htmz seems fine as a library. It feels kinda powerful for its tiny size. (But really it’s the browser that’s doing the heavy lifting!) Nonetheless, there are limitations.

What are the limitations?

The main direct limitation is having only one destination per response. However, this can be fixed by writing an extension. ;)

A more general but classic limitation is the request-response model. The Web 1.0 model, and the baggage that comes with it. Like a roundtrip delay on every interaction, a browser history entry on every click, etc.

The Web 1.0 model might also mean putting more UI logic in the web server. This can be a good thing or a bad thing, as it could either lead to consolidation or fragmentation of UI logic, which respectively decreases or increases complexity. It really depends on your goal and style.

=htmz>