Reactivity without the Framework

A tiny ~2kb library for building reactive interfaces in native JavaScript

But why? Get Started

ArrowJS is an experimental tool for programming reactive interfaces using native JavaScript. It’s not really a framework, but not less powerful than a

framework either.

At its core — ArrowJS is an admission that while we developers were busy falling in love with fancy UI frameworks, JavaScript itself got good — like *really* good.

Overview

If JavaScript is so good, then what does a tool like Arrow bring to the table? So glad you asked. Arrow has 2 primary features:

- Observable data.
- Declarative/Reactive DOM rendering.

For many applications, these two features are all you need to build delightful and complex user interfaces. Need state management? Use a module's scope. Need components? Use functions. Need routing? The web platform already does this pretty well 😉.

Additionally, Arrow boasts a few more important talking points:

- Zero dependencies.
- No build tools required (or even suggested).
- Less than 3KB min+gzip. (22x smaller than this itty bitty gif → )

Got time for a quick example? Great.

```
import { reactive, html } from '@arrow-js/core'

const data = reactive({
  clicks: 0
});

html`
  <button @click="${() => data.clicks++}">
    Fired ${() => data.clicks} arrows
  </button>
`
```

Fired 0 arrows

Key Commitments

Commitment to JavaScript

Arrow relies heavily on modern features of JavaScript such as template literals, modules (think `import` and `export`), and Proxies. For example, you'll immediately notice that Arrow does not have a special template "language" like so many other frameworks. Instead it relies on template literals (tick marks `` ` ``) — specifically tagged template literals — to interpolate expressions and render DOM elements. For example:

```
const third = 'Third';

html`
  <ul>
```

```
    <li>First</li>
    <li>Second</li>
    <li>${third}</li>
  </ul>
`
```

- First
- Second
- Third

We go in depth on templates in the docs, but a key concept to understand here is that template literals, and tagged template literals, are **native features of JavaScript**.

Why does this matter? Well for one it makes Arrow fast — most of the parsing is done using language-level features. More importantly, however, learning Arrow is mostly learning how to use modern native JavaScript to create UI systems, so the concepts here are portable.

Already fancy yourself a great JavaScript developer? Great! Then learning Arrow won't take you any time at all.

Commitment to no build tools

Build tools can be useful. Arrow itself is written in TypeScript so it necessitates a build script to compile, but while there is no restriction against using a build tool, Arrow *will never require one*. Arrow removes the need for complex operations that are best left to compilers, like converting templates to render functions. It does this by making some assumptions:

- It's ok to ship modern JS (no IE support)
- You’re writing HTML (not native voodoo)

It will always be good and right to pull in Arrow from a CDN and start building your project right away.

Commitment to performance

Arrow is *fast.* Downloading, booting, and patching are all fast. In fact, you can generally expect on-par-or-better performance than its bigger JS framework counterparts. Arrow will always be a guilt-free choice for those under a performance budget.

Commitment to Open Source

Arrow was created by me, Justin Schroeder. It is Open Source. It will always be Open Source. My hope is this project helps reframe developer's expectations of "native" JavaScript.

Get Started with ArrowJS