HTML FORMS: MORE POWERFUL THAN YOU THINK

924 words Wed 17th December, 2025

> *"Does anyone ACTUALLY like HTML forms?"*

Rich Harris of Svelte fame asked during his closing keynote at performance.now() 2025 this year.

I do. I fucking love HTML forms.

Photo credit: Richard Theemling

To be fair to Rich, he's a good chap and he was primarily questioning the integration between client and server. And *of course* it was rhetorical, but I still raised my hand 🙋🏻‍♂️

Forms were standardised in HTML 2.0 a whopping 30 years ago. They are foundational to the web and because of this deep integration with the platform, they have a lot of powerful features that many developers are unaware of.

THE (LESSER-KNOWN) BASICS

Let's start simple: you can switch the `method` to `get` to make regular page navigations. Most people know this but too often overlook it. It's great for saving state in the URL, making it shareable and bookmarkable — think things like search forms, facet-based filtering etc.

You can have different buttons submit the same serialised data to different endpoints by adding `formaction` and/or `formmethod`.

For multiple submit buttons within the same form, you can add a `name` and `value` just like any `<input>` to distinguish which button was pressed.

You can even have buttons and input fields reference forms FROM THE OUTSIDE or even re-associated one forms' inputs/buttons with another via the `form` attribute!

You can have them submit their data to an `<iframe>` via the `target` and `formtarget` attributes — this is something we did before AJAX existed.

Combining these properties, I was able to create this user interface where one simple form provided three functions:

1. Preview iframe, via `formmethod="get" formtarget="preview-iframe" formaction="/preview"`
2. Save settings button, via `formmethod="post" formaction="/save-settings"`
3. Download image button, via `formmethod="post" formaction="/download-image"`

To create this UI:

There was some very minimal extra progressive-enhancement to improve the user experience:

I added an `<auto-submit>` custom element to submit the form whenever any descendant input changes so the iframe updates automatically.

I enhanced the iframe with View Transitions to provide a smooth transition when updating the preview.

And the excerpt highlight range is submitted as hidden inputs, with the persistent highlight powered by the CSS Custom Highlight API.

INPUT CUSTOMISATION

The following are features you should be using to improve the user experience of your forms, these are agnostic of whether you're using client-side JavaScript frameworks and their associated form libaries, or just plain old HTML.

inputmode let's you trigger different virtual keyboards on mobile devices, e.g. numeric, tel, email etc.

You have to be careful with autofocus but it can be useful for things like <dialog> where you might want to focus on an input as soon as it opens, for example modal-based typeahead search.

autocomplete helps browsers provide better autofill suggestions, hopefully eliminating a lot of tedious typing for your users.

autocapitalize can be used to control whether the first letter of sentences, words or characters are capitalised on mobile devices.

spellcheck can be used to enable or disable spell checking on text inputs and textareas.

datalist can be used to provide a list of predefined options for an <input> while still allowing freeform input.

capture can be used on mobiles to directly open the front or rear camera for image or video uploads:

```
<input type="file" capture="user" accept="video/*" />
<input type="file" capture="environment" accept="image/*" />
```

CLIENT-SIDE VALIDATION

Alongside the input types like email, url and number, there's a bunch of built-in validation available via attributes like required, minlength / maxlength and pattern — but did you know you can extend this by providing custom validation messages with setCustomValidity()?

In some cases, you may want to use novalidate on your <form> or formnovalidate on the <button> and <input> elements to disable validation entirely. The latter can be useful for "save for later" buttons that shouldn't require all fields to be valid.

STYLING

There’s :valid and :invalid CSS selectors and their :user-valid :user-invalid counterparts, which when combined with :has give you powerful styling options, e.g.

```
form:has(:invalid) [type=submit] {
  background-color: #ccc;
  color: #666;
  cursor: not-allowed;
}
```

`field-sizing: content` lets us adjust an input's width or height based on the content.

MORE RECENT ADDITIONS

With the Navigation API, they become easier than ever to progressively enhance. The following few lines add an `aria-busy` attribute to forms while they are submitting:

```
window.navigation?.addEventListener("navigate", (event) => {
  const formElement = event.sourceElement?.closest("form");
  const buttons = formElement?.querySelectorAll("button, input[type='submit']");
  if (buttons?.length) requestAnimationFrame(() => buttons.forEach(button => button.setAttribute("ar
});
```

You can serialise them easily in JavaScript with the `FormData` API, which can be used with `fetch`, `URLSearchParams` and wherever else you might want to access the values. In the past, I've used this to build lightweight custom elements for conditional rendering of form fields based on other field values:

```
class ConditionalFormField extends HTMLElement {
  connectedCallback() {
    this.form = this.closest("form")
    form.addEventListener("input", this)
    form.addEventListener("change", this)
    // perform initial evaluation on connect, handling existing input from SSR
    this.handleEvent()
  }

  handleEvent() {
    // serialize our form
    const formData = new FormData(this.form)
    // test whether our form meets our condition in the format { "field-name": "expected-value", ..
    const conditionMet = Object.keys(this.condition).every(key => formData.get(key) === this.conditi
    // toggle our hidden attribute and disable child inputs (to prevent serialization) based on whet
    requestAnimationFrame(() => {
      this.toggleAttribute("hidden", !conditionMet)
      this.querySelectorAll("input, select, textarea").forEach((input) => {
        input.toggleAttribute("disabled", !conditionMet)
      })
    })
  }
  get condition() { return this.cachedCondition ??= JSON.parse(this.getAttribute("data-condition"))
}

customElements.define('conditional-form-field', ConditionalFormField)
```

Next time you're building a form, consider leaning into the platform features available to you first.

Yes, you'll regularly need JavaScript to enhance the experience, but the less we rely on it for fundamental features, the better our performance, reliability and maintainability will be.

Previously:

YOU DON'T BECOME A UNICORN BY ACTING LIKE ONE

1123 words Tue 16th December, 2025

Next up:

DATABASE VIEWS FOR DEBUGGING

302 words Sun 14th December, 2025