



Making HTML More Expressive

R. Mark Volkmann

Object Computing, Inc.



<https://objectcomputing.com>



mark@objectcomputing.com



[@mark_volkmann](https://twitter.com/mark_volkmann)



OBJECT COMPUTING
YOUR OUTCOMES ENGINEERED

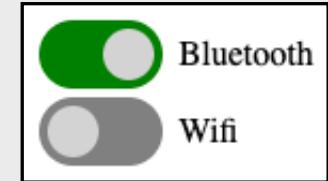
Approaches

- There are many ways to make HTML more expressive
 - **template element** and a bit of JavaScript
 - **web components** (vanilla, Lit, and Shoelace)
 - **new attributes on HTML elements** (Alpine and htmx)
- Let's review each of these



Quotes

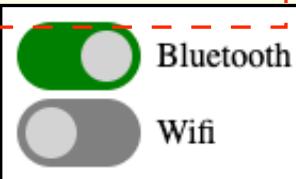
- “Life is like a box of chocolates.”
 - Forrest Gump
- “A checkbox is like a toggle switch.”
 - Me
- “You never know what you’re gonna get.”
 - Forrest Gump
- “You will get a form component that represents a Boolean state.”
 - Me
- “A toggle switch is better than a box of chocolates.”
 - Me



template Element



```
<!DOCTYPE html>                                public/index.html
<html>
  <head>
    <title>Template Toggle</title>
    <link rel="stylesheet" href="./toggle-switch.css" />
    <script defer src="./toggle-switch.js"></script>
  </head>
  <body>
    <template id="toggle-switch-template">
      <label class="toggle-switch">
        <input class="thumb" type="checkbox" />
        <div class="switch"></div>
        <span class="label"></span>
      </label>
    </template>
    <div id="target1"></div>
    <div id="target2"></div>
  </body>
</html>
```



The code defines a template for a toggle switch. It includes a label with a checkbox input and a surrounding div. A span is used for the label text. The template is then used in the body to create two separate divs, each containing a toggle switch. The first switch is labeled 'Bluetooth' and the second is labeled 'Wifi'.

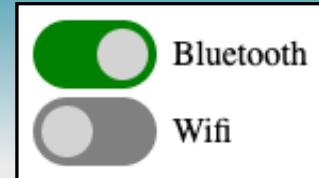
```
public/toggle-switch.js
```

```
function toggleSwitch(targetId, label, checked, callback) {
  const template = document.getElementById('toggle-switch-template');
  const clone = template.content.cloneNode(true); // deep
  const labelSpan = clone.querySelector('.label');
  labelSpan.textContent = label;
  const input = clone.querySelector('[type=checkbox]');
  input.checked = checked;
  input.addEventListener('change', callback);

  const target = document.getElementById(targetId);
  target.appendChild(clone);
}

toggleSwitch('target1', 'Bluetooth', true, e => {
  console.log('Bluetooth:', e.target.checked);
});
toggleSwitch('target2', 'Wifi', false, e => {
  console.log('Wifi:', e.target.checked);
});
```

Toggle Switch CSS ...



surrounds entire component

```
label.toggle-switch {  
  --use-padding: var(--padding, 0.25rem);  
  --use-switch-height: var(--switch-height, 2rem);  
  --switch-width: calc(var(--use-switch-height) * 1.8);  
  --thumb-size: calc(  
    var(--use-switch-height) -  
    var(--use-padding) * 2  
  );  
  --use-transition-duration: var(--transition-duration, 0.3s);  
  
  display: inline-flex;  
  align-items: center;  
  gap: 0.5rem;  
  color: var(--label-color, black);  
  cursor: pointer;  
  
  & .switch {  
    background-color: var(--off-bg, gray);  
    border-radius: calc(var(--use-switch-height) / 2);  
    height: var(--use-switch-height);  
    position: relative;  
    transition: background var(--use-transition-duration);  
    width: var(--switch-width);  
  }  
}
```

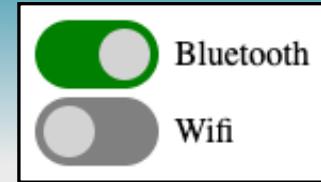
public/toggle-switch.css

configurable via CSS variables
(aka custom properties)

```
/* This renders the thumb.  
 It is absolutely positioned inside the switch. */  
&:before {  
  content: '';  
  background-color: var(--thumb-bg, lightgray);  
  border-radius: 50%;  
  transition: left var(--use-transition-duration);  
  
  position: absolute;  
  top: var(--use-padding);  
  left: var(--use-padding); /* unchecked position */  
  
  width: var(--thumb-size);  
  height: var(--thumb-size);  
}
```

content created by CSS

... Toggle Switch CSS



```
/* The thumb is a preceding sibling of the switch. */
.thumb:checked + & {
  background-color: var(--on-bg, green);
  &:before {
    /* checked position */
    left: calc(
      var(--switch-width) -
      var(--use-switch-height) +
      var(--use-padding)
    );
  }
}

/* The thumb input is only used to hold the checked state.
   It is not rendered. Instead, the before content of the switch
   is used for the thumb. */
& .thumb {
  position: absolute;
  visibility: hidden;
}
}
```

not visible

Web Components



- Define **custom HTML elements** that can be used just like standard HTML elements
 - names must be all lower-case and contain at least one hyphen
 - tags cannot be self-closing ex. `<oci-toggle></oci-toggle>`, not `<oci-toggle />`
- Can be used in any web page, with **any web framework**, and in Markdown files
- Requires **a bit more effort** than implementing components using a framework like Svelte
 - worthwhile for components that may someday be used in multiple apps written using multiple frameworks
- **Standards-based:** Custom Elements, Shadow DOM, ES Modules, and HTML Templates

WC Lifecycle Methods



- **constructor**
 - called when an instance is initially created
- **connectedCallback**
 - called after an instance is added to DOM
- **attributeChangedCallback**
 - called when the value of any “observed” attribute changes
 - specify with

```
static get observedAttributes() { return ['name1', 'name2', ...]; }
```
- **disconnectedCallback**
 - called after an instance is removed from DOM

Vanilla Web Component ...



must extend `HTMLElement`

```
export class ToggleSwitchWC extends HTMLElement {    src/toggle-switch-wc.js
  constructor() {
    super();

    const label = this.getAttribute('label');
    const checked = this.getAttribute('checked');

    this.attachShadow({mode: 'open'});
    const root = this.shadowRoot;
    root.innerHTML =
      <style>${ToggleSwitchWC.styles}</style>
      <label>
        <input class="thumb" type="checkbox" checked=${checked}/>
        <div class="switch"></div>
        <span class="label">${label}</span>
      </label>
    ;

    const checkbox = root.querySelector('input');
    checkbox?.addEventListener('change', this.handleChange.bind(this));
  }
}
```



```
handleChange(event: Event) {
  const root = this.shadowRoot!;

  const {checked} = event.target;
  const newEvent = new CustomEvent(
    'toggle',
    {detail: {checked}}
  );
  this.dispatchEvent(newEvent);
}

static styles = `...same as before...`;

customElements.define(
  'toggle-switch-wc',
  ToggleSwitchWC
);
```

setting `innerHTML` removes the need to use low-level
DOM methods like `createElement` and `appendChild`

Alternative

This approach supports updating UI when attribute values are changed by calling **setAttribute** OR **removeAttribute**.

```
export class ToggleSwitchWC extends HTMLElement {
  constructor() {
    super();
    src/toggle-switch-wc.js

    this.attachShadow({mode: 'open'});
    const root = this.shadowRoot;
    root.innerHTML =
      <style>${ToggleSwitchWC.styles}</style>
      <label>
        <input class="thumb" type="checkbox" />
        <div class="switch"></div>
        <span class="label"></span>
      </label>
    ;
    <img alt="A green toggle switch icon with the text 'Web Component' next to it." data-bbox="205 590 365 635"/>
    const checkbox = root.querySelector('input');
    checkbox?.addEventListener(
      'change',
      this.handleChange.bind(this)
    );
  }

  static get observedAttributes() {
    return ['checked', 'label'];
  }
}
```

```
attributeChangedCallback(
  name: string,
  _: string | null,
  newValue: string | null
) {
  const root = this.shadowRoot!;
  if (name === 'checked') {
    const checkbox = root.querySelector('input');
    if (checkbox) {
      if (newValue === null) {
        checkbox.removeAttribute('checked');
      } else {
        checkbox.setAttribute('checked', 'checked');
      }
    }
  } else if (name === 'label') {
    const span = root.querySelector('.label');
    if (span) span.textContent = newValue;
  }
}

handleChange(event: Event) {
  const root = this.shadowRoot!;
  const {checked} = event.target;
  const newEvent = new CustomEvent(
    'toggle',
    {detail: {checked}}
  );
  this.dispatchEvent(newEvent);
}

static styles = `...same as before...`;

customElements.define('toggle-switch-wc', ToggleSwitchWC);
```

... Vanilla Web Component



```
<!DOCTYPE html>
```

```
<html>
  <head>
    <style>
      toggle-switch-wc {
        --label-color: purple;
        --off-bg: red;
        --on-bg: blue;
        --padding: 1rem;
        --switch-height: 4rem;
        --thumb-bg: yellow;
        --transition-duration: 1s;
      }
    </style>
  <script defer
    src="https://cdn.jsdelivr.net/npm/alpinejs@3.x.x/dist/cdn.min.js"></script>
  <script type="module" src="/src/toggle-switch-wc.js"></script>
  <script>
```

```
    function handleToggle(event) {
      const {checked} = event.detail;
      console.log('checked =', checked);
    }
  </script>
</head>
```

public/index.html

demonstrates setting all the CSS custom properties used in the component CSS

using Alpine for event handling (@toggle)



a real app would do more here

```
<body>
  <toggle-switch-wc
    label="Web Component"
    checked
    @toggle="handleToggle($event)"
  ></toggle-switch-wc>
</body>
</html>
```

@toggle and \$event are specific to Alpine

Lit

<https://lit.dev/>



- Lit components
 - are native web components
 - can be used everywhere web components can be used
 - can be implemented in JavaScript (no build step) or TypeScript
 - require loading the Lit library
 - use `html` and `css` tagged template literals
- Decorator syntax requires TypeScript
- Provides a large amount of functionality
 - goes well beyond just implementing web components
 - can be used as a replacement for SPA frameworks like React

Lit Web Component

```
src/toggle-switch-lit.js

import {css, html, LitElement} from 'lit';
import {customElement, property} from 'lit/decorators.js';

@customElement('toggle-switch-lit')
export class ToggleSwitchLit extends LitElement {
  @property() label = '';
  @property({type: Boolean}) checked = false;

  render() {
    return html`
      <input
        class="thumb"
        type="checkbox"
        ?checked=${this.checked}
        @change=${this.handleChange}
      />
      <div class="switch"></div>
      <span class="label">${this.label}</span>
    `;
  }
}

must extend LitElement
```

?checked and @change are specific to Lit

```
handleChange(event: Event) {
  const {checked} = event.target;
  const newEvent = new CustomEvent(
    'toggle',
    {detail: {checked}}
  );
  this.dispatchEvent(newEvent);
}

static styles = css`...same as before...`;
```

HTML to use this would be similar to that for the vanilla web component.



- Large collection of open source web components that can be used with all web frameworks or JavaScript with no framework
- Built with Lit
- Can use from CDN or install from npm
- <https://shoelace.style/>

Shoelace sl-switch

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Shoelace Demo</title>
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/@shoelace-style/shoelace@2.14.0/cdn/themes/light.css"
    />
    <script
      defer
      src="https://cdn.jsdelivr.net/npm/alpinejs@3.x.x/dist/cdn.min.js"></script>
    <script
      type="module"
      src="https://cdn.jsdelivr.net/npm/@shoelace-style/shoelace@2.14.0/cdn/shoelace-autoloader.js"></script>
    <script>
      function handleChange(event) {
        const {checked} = event.target;
        console.log('checked =', checked);
      }
    </script>
  </head>
  <body x-data>
    <div>
      <sl-switch size="large" checked @sl-change="handleChange">
        Shoelace
      </sl-switch>
    </div>
  </body>
</html>
```

public/index.html

using Alpine for event handling (@sl-change)

required to activate Alpine



Alpine



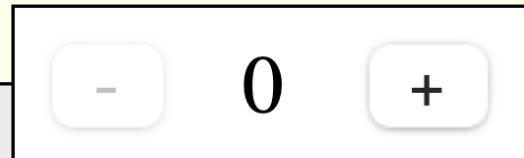
- “Lightweight JavaScript framework that uses **custom HTML attributes** to add dynamic behavior”
- “Offers you the **reactive and declarative** nature of big frameworks like Vue or React at a much lower cost”
- **Small library**
- **No build process**
- Provides a large number of “**directives**” that are applied as HTML attributes
 - ex. **x-bind**, **x-data**, **x-for**, **x-if**, **x-model**, **x-on**, **x-show**, and **x-text**

shorthand is :

shorthand is @

Alpine Counter

```
<!DOCTYPE html>
<html>
  <head>
    <title>Alpine Counter</title>
    <script
      defer
      src="https://cdn.jsdelivr.net/npm/alpinejs@3.x.x/dist/cdn.min.js"></script>
  </head>
  <body>
    <div style="display: flex; gap: 1rem" x-data="{count: 0}">
      <button @click="count--" :disabled="count === 0">-</button>
      <span x-text="count"></span>
      <button @click="count++" :disabled="count === 10">+</button>
    </div>
  </body>
</html>
```

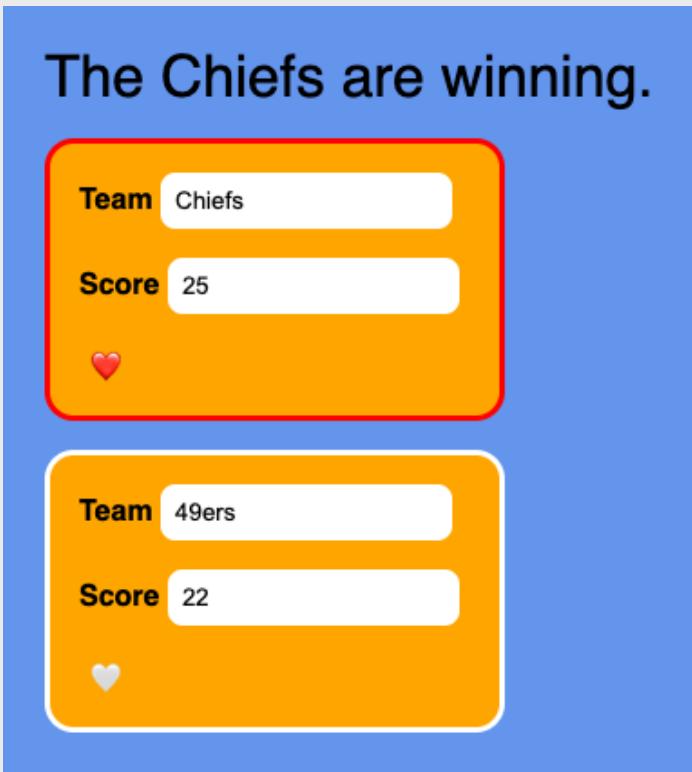


x-data is the Key!

- Serves two purposes
 - declares reactive properties
 - activates use of Alpine →

Forgetting this is the #1 source of issues because other directives are ignored if not on or inside an element with this.
- Can apply to multiple elements
 - each defines a scope
 - scopes can be nested

Alpine Score Keeper



```
score-keeper.css
```

```
body {  
    background-color: cornflowerblue;  
    font-family: sans-serif;  
    font-size: 1rem;  
    padding: 1rem;  
}  
  
button {  
    background-color: transparent;  
    border: none;  
}  
  
.column {  
    display: flex;  
    flex-direction: column;  
    align-items: start;  
    gap: 1rem;  
}  
  
input {  
    border: none;  
    border-radius: 0.5rem;  
    padding: 0.5rem;  
}  
  
label {  
    font-weight: bold;  
}  
  
#report {  
    font-size: 2rem;  
}  
  
.team {  
    background-color: orange;  
    border: 3px solid white;  
    border-radius: 1rem;  
    padding: 1rem;  
    width: 13.5rem;  
}
```

... Alpine Score Keeper

```

<html>
  <head>
    <title>Alpine Score Keeper</title>
    <link rel="stylesheet" href="score-keeper.css" />
    <script
      defer
      src="https://cdn.jsdelivr.net/npm/
        alpinejs@3.x.x/dist/cdn.min.js"
    ></script>
    <script>
      const getData = () => ({
        dislikeColor: "white",
        likeColor: "red",
        score1: 25,
        score2: 22,
        team1: "Chiefs",
        team2: "49ers",
        // This is like a computed property.
        report() {
          const s1 = Number(this.score1);
          const s2 = Number(this.score2);
          return s1 > s2
            ? `The ${this.team1} are winning.`
            : s2 > s1
            ? `The ${this.team2} are winning.`
            : "The score is tied.";
        },
      });
    </script>
  </head>

```

score-keeper.html

outer **x-data scope**

inner **x-data scopes**

```

<body>
  <main class="column" x-data="getData">
    <div id="report" x-text="report"></div>
    <section
      class="column team"
      :style="`border-color: ${like ? likeColor : dislikeColor}`"
      x-data="{like: false}"
    >
      <label>Team <input type="text" x-model="team1" /></label>
      <label>Score <input type="number" x-model="score1" /></label>
      <button @click="like = !like" x-text="like ? '❤️' : '🤍'"></button>
    </section>
    <!-- We could avoid this repetition by creating a web component. -->
    <section
      class="column team"
      :style="`border-color: ${like ? likeColor : dislikeColor}`"
      x-data="{like: false}"
    >
      <label>Team <input type="text" x-model="team2" /></label>
      <label>Score <input type="number" x-model="score2" /></label>
      <button @click="like = !like" x-text="like ? '❤️' : '🤍'"></button>
    </section>
  </main>
</body>
</html>

```

The Chiefs are winning.

Team	Chiefs
Score	25
❤️	

Team	49ers
Score	22
🤍	

htmx Overview



- Client-side JavaScript library for implementing hypermedia-driven applications (HDAs)
 - adds support for new HTML attributes that make HTML more expressive
 - uses endpoints that return HTML rather than JSON
 - free and open-source
- Sponsored by 19 companies (as of Feb. 2024)
 - including GitHub and JetBrains
- <https://htmx.org/>

htmx Improves HTML

HTML	htmx
only anchor and <code>form</code> elements trigger HTTP requests	any element can trigger HTTP requests
only triggered by clicking an anchor or submitting a form	any event can trigger
only GET and POST requests are sent	any verb can be used, including PUT, PATCH, and DELETE
response causes a full page refresh	response can be inserted into current page

SPA Frameworks vs htmx

- **SPA approach**

- HTTP responses contain JSON
- client-side code parses JSON
- client-side code uses result to build DOM
- framework code inserts new DOM into current page



- **htmx approach**

- HTTP responses contain HTML
- browser automatically builds DOM from HTML
- htmx inserts new DOM into current page



faster due to elimination of
JSON generation and parsing



HATEOAS



- Stands for Hypermedia As The Engine Of Application State
- Major focus of htmx
- **Hypermedia:** any data format that can describe branching from one “media” (ex. a document) to another
- HTML is a kind of hypermedia, but JSON is not
- **Hypermedia client:** software that understands and renders a hypermedia format, such as web browsers with HTML
 - no custom client-side code is required



htmx Tech Stacks ...



- Can use any programming language and framework that can implement an **HTTP server** whose endpoints return **HTML responses**
- Referred to as “Hypermedia On Whatever you’d Like” (**HOWL**)
- **Good choices make it easy to**
 - create new endpoints for any HTTP verb
 - specify type checking and validation of request data
 - get request data from headers, path parameters, query parameters, and bodies
 - send HTTP responses that include headers and bodies that contain text or HTML



... Tech Stacks

- **Good choices have tooling that supports**
 - **fast server startup** with no build process or a simple one
 - **automatic server restarts** after source code changes are detected
 - **good HTML templating** support (such as JSX) or my npm package js2htmlstr without relying on string concatenation
 - **syntax highlighting** of HTML in code editors



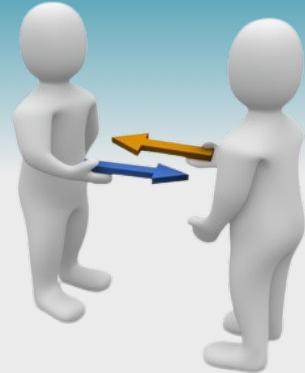
htmx Basics



- Add htmx attributes to elements that trigger HTTP requests
- Specify events that trigger the request
 - **hx-trigger** comma-separated list of event names with optional modifiers
- Specify HTTP verb to use and endpoint URL
 - **hx-get**, **hx-post**, **hx-put**, **hx-patch**, and **hx-delete**
- Specify element where response HTML will go
 - **hx-target** can be CSS selector and/or use several keywords; defaults to **this**
- Specify where to place HTML relative to target
 - **hx-swap** see next slide

All elements have a **default trigger**.
form elements trigger on **submit**.
input, **textarea**, and **select** elements trigger on **change**.
All other elements trigger on **click**.

hx-swap



Assume **hx-target** refers to the **ul** element.

Options to
insert content

beforebegin

afterbegin

beforeend

afterend

```
<p>before list</p>
<ul>
  <li>Red</li>
  <li>Green</li>
  <li>Blue</li>
</ul>
<p>after list</p>
```

Options to
replace content

outerHTML

innerHTML (default)

Options that do not
use response HTML

delete

removes target element

none

leaves target element as-is

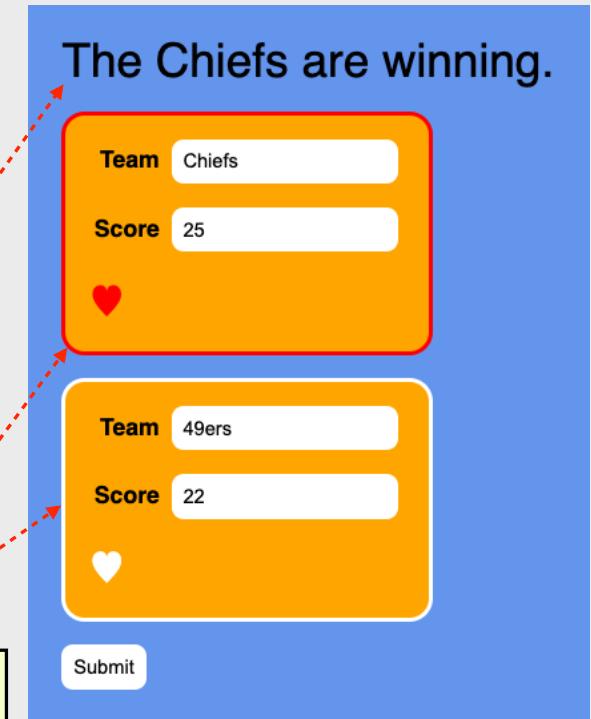
htmx Score Keeper

- Let's implement the same app again using htmx
- Scores are sent to a server
 - requirement for many applications

```
<html>
  <head>
    <title>htmx Score Keeper</title>
    <link rel="stylesheet" href="styles.css" />
    <script src="https://unpkg.com/htmx.org@1.9.10"></script>
  </head>
  <body>
    <div hx-trigger="load" hx-get="/report" id="report"></div>
    <form class="column" hx-post="/update" hx-target="#report">
      <div hx-trigger="load" hx-get="/team/1" hx-target="this"></div>
      <div hx-trigger="load" hx-get="/team/2" hx-target="this"></div>
      <button>Submit</button>
    </form>
  </body>
</html>
```

index.html

In elements with an ancestor that triggers HTTP requests, **hx-target** defaults to that ancestor. This is why **hx-target="this"** is required on these **div** elements.



htmx Score Keeper - CSS

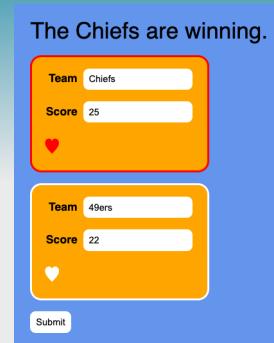
```
body {  
    background-color: cornflowerblue;  
    font-family: sans-serif;  
    font-size: 1rem;  
    padding: 1rem;  
}  
  
button {  
    background-color: white;  
    border: none;  
    border-radius: 0.5rem;  
    padding: 0.5rem;  
}  
  
.column {  
    display: flex;  
    flex-direction: column;  
    align-items: start;  
    gap: 1rem;  
}  
  
input {  
    border: none;  
    border-radius: 0.5rem;  
    padding: 0.5rem;  
}
```

styles.css mostly same as CSS for Alpine version

```
input[type='checkbox'] {  
    display: none;  
}  
  
input[type='checkbox'] + label {  
    color: white;  
    font-size: 1.5rem;  
}  
  
input[type='checkbox']:checked + label {  
    color: red;  
}  
  
label > div {  
    display: inline-block;  
    font-weight: bold;  
    margin-right: 0.5rem;  
    text-align: right;  
    width: 3rem;  
}
```

see red dashed box on next slide

```
#report {  
    font-size: 2rem;  
    margin-bottom: 1rem;  
}  
  
.team {  
    background-color: orange;  
    border: 3px solid white;  
    border-radius: 1rem;  
    padding: 1rem;  
    width: 13.5rem;  
}
```



htmx Score Keeper - TS ...

```
server.tsx
import {type Context, Hono} from 'hono';
import {serveStatic} from 'hono/bun';

type Team = {
  name: string;
  score: number;
  like: boolean;
};
const team1: Team = {
  name: 'Chiefs',
  score: 25,
  like: true
};
const team2: Team = {
  name: '49ers',
  score: 22,
  like: false
};

function report(): string {
  return team1.score > team2.score
    ? `The ${team1.name} are winning.`
    : team2.score > team1.score
    ? `The ${team2.name} are winning.`
    : 'The score is tied.';
}
```

Checkbox is used to hold state, but is not displayed.

```
function teamHtml(number: string): JSX.Element {
  const team = number === '1' ? team1 : team2;
  const borderColor = team.like ? 'red' : 'white';
  return (
    <section class="column team"
      style={`border-color: ${borderColor}`}
    >
      <label>
        <div>Team</div>
        <input type="text" name={'team' + number}
          required value={team.name} />
      </label>
      <label>
        <div>Score</div>
        <input type="number" name={'score' + number}
          required value={team.score} />
      </label>
      <input
        type="checkbox"
        id={'like' + number}
        checked={team.like}
        hx-get={'/toggle-like/' + number}
      />
      <label for={'like' + number}>&hearts;</label>
    </section>
  );
}
```

Clicking a label associated with a checkbox toggles it.



... htmx Score Keeper - TS

```
const app = new Hono();

// Serve static files from public directory.
app.use('*', serveStatic({root: './public'})); ← index.html and styles.css

app.get('/team/:number', (c: Context) => {
  const number = c.req.param('number');
  return c.html(teamHtml(number));
});

app.get('/report', (c: Context) => c.text(report()));

app.get('/toggle-like/:number', (c: Context) => {
  const number = c.req.param('number');
  const team = number === '1' ? team1 : team2;
  team.like = !team.like;
  return c.html(teamHtml(number));
});

app.post('/update', async (c: Context) => {
  const formData = await c.req.formData();
  team1.name = formData.get('team1') as string;
  team2.name = formData.get('team2') as string;
  team1.score = Number(formData.get('score1'));
  team2.score = Number(formData.get('score2'));
  return c.text(report());
});

export default app;
```



Resources

- **Web components MDN page** -
https://developer.mozilla.org/en-US/docs/Web/API/Web_components
- **Lit** home page - <https://lit.dev>
- **Shoelace** home page - <https://shoelace.style>
- **Alpine** home page - <https://alpinejs.dev>
- **htmx** home page - <https://htmx.org>
- **My blog** - <https://mvolkmann.github.io/blog>
- **“Hypermedia Systems” book** - <https://hypermedia.systems/>



Wrap Up

- We have seen many ways to make HTML more expressive
- All of these can be used without a build process ... just open an HTML file in a web browser
- Large frameworks with sizable learning curves are not required to build fully functional, highly interactive web applications

