The Pragmatic Programmers

Pragmatic exPress

# Server-Driven Web Apps with htmx

## Any Language, Less Code, Simpler Code

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Copyright © The Pragmatic Programmers, LLC.

CHAPTER 1

# Jumping In

Let's jump in and learn the basics of htmx in order to get a taste of how its approach to web development differs from those you have already experienced. First we'll choose a tech stack. Then we'll learn about the most commonly used htmx attributes. Finally, we'll implement two small apps using htmx.

## Choosing a Tech Stack

Before you can implement a web app using htmx, you need to choose a tech stack.

The server side of htmx web applications can be implemented with any programming language that supports HTML templating (e.g. JSX) and has an HTTP server library. This is referred to as "Hypermedia On Whatever you'd Like" (HOWL). Popular choices include JavaScript, Python, and Go, but you are not limited to these.

In the next chapter you'll learn how to evaluate the suitability of a particular tech stack for use with htmx. For now we will just pick one so we can dive into our first code example.

This book uses the following:

- Bun,[1] a JavaScript runtime, package manager, bundler, and test runner
- TypeScript,[2] a superset of JavaScript that adds support for types
- Hono,[3] a TypeScript library for implementing HTTP servers

The main reason we chose a JavaScript-based stack is that many readers are web developers who are already familiar with JavaScript.

1. https://mvolkmann.github.io/blog/topics/#/blog/bun
2. https://objectcomputing.com/resources/publications/sett/
3. https://mvolkmann.github.io/blog/topics/#/blog/hono

• Click HERE to purchase this book now. discuss

Don't despair if these are not choices you would make. The stars of the show here are HTML, CSS, and htmx. The use of htmx with HTML is what matters most and that is the focus of this book. You can choose a different tech stack and still benefit from what you learn here.

**Joe asks:**

### Why would you choose to use JavaScript when htmx opens the possibility to use any programming language?

The Bun JavaScript runtime delivers very good performance, at least in comparison to Python. Bun also provides a great way to generate HTML using JSX. Using TypeScript, a superset of JavaScript, provides type checking. When combined with the Hono server library, endpoints can be implemented in concise code. A new endpoint can be defined by editing a single source file. Some frameworks (like Django) require editing three files, one for the endpoint code, one for an HTML template, and one to register a URL for the endpoint.

## Using htmx Attributes

Htmx provides a new set of HTML attributes that make HTML more expressive. The htmx library processes these attributes. Some of them cause HTTP requests to be sent to endpoints that return HTML which is inserted into the DOM. Without htmx, doing this requires writing custom JavaScript code.

Any event on any HTML element can trigger any kind of HTTP request (GET, POST, PUT, PATCH, or DELETE) and the response will not result in a full page refresh. All this is done without writing any custom client-side JavaScript code.

Currently htmx defines 36 attributes, but a small subset of them are commonly used. Let's discuss those commonly used attributes, which answer the following questions:

*What events trigger a request?*
*A mouse click, a form submission, other events?*

The `hx-trigger` attribute specifies the kinds of events that will trigger a request.

*What kind of request should be sent: GET, POST, PUT, PATCH or DELETE? And where should the request be sent?*

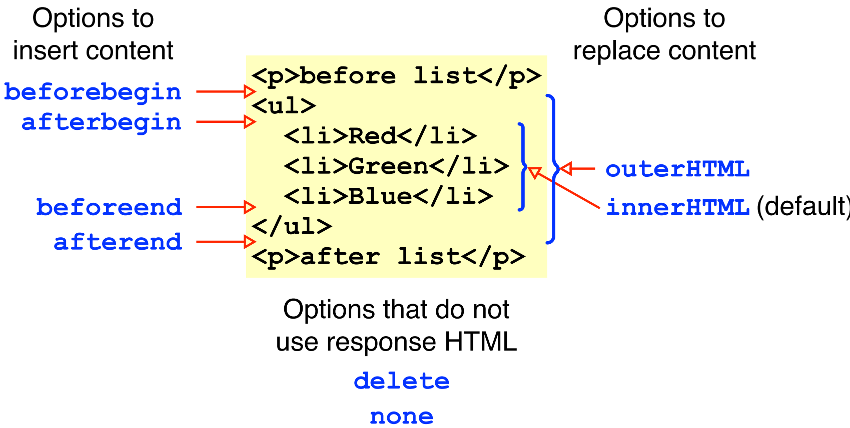
The hx-get, hx-post, hx-put, hx-patch, and hx-delete attributes describe both the kind of request to be sent and the URL where it will be sent.

*When the endpoint returns HTML, what element should receive it?*

The hx-target attribute indicates the intended destination (target) of the returned HTML.

*How should the new HTML be placed relative to the target element?*

The hx-swap attribute details exactly how the returned HTML will be placed relative to the target. The options are described in the diagram below.




## Creating Your First Project

Let's start simple to get a feel for using htmx. Open a terminal window, and install Bun by entering the following command:

~~~ console $ curl -fsSL https://bun.sh/install | bash ~~~

Windows users can use WSL to enter this curl command. Another option is to enter the command powershell -c "irm bun.sh/install.ps1|iex". Yet another is to install the Chocolatey[4] package manager for Windows and enter choco install bun.

When the install is finished, cd to the directory where the project will be created, and enter bunx create-hono to create the project. After the "Target directory" prompt, enter a project name like "htmx-demo". The next prompt will be

4. https://chocolatey.org

• Click HERE to purchase this book now. discuss
~~~

"Which template do you want to use?"; select "bun". The next prompt will be "Do you want to install project dependencies?"; enter "Y". The next prompt will be "Which package manager do you want to use?"; select "bun". Now cd to the new project directory which will contain the following:

- README.md – contains instructions on running the project
- package.json – describes project dependencies and defines a script for running the project
- tsconfig.json – configures the use of TypeScript
- .gitignore – prevents the node_modules directory from being committed
- src/index.ts – implements a Hono HTTP server and defines the "GET /" endpoint

Enter bun install. This creates the node_modules directory and installs all the required dependencies there. Start a local server by entering bun dev.

Now go to a web browser, and browse localhost:3000. You'll see that it renders "Hello Hono!".

Now that we have a default project, let's modify it to use htmx. Start by renaming the file src/index.ts to src/server.tsx. The .tsx file extension is a convention that enables using JSX to generate HTML.

**Using JSX**

The project we are building here will be used as a template for all the other projects we will build later. While this project doesn't actually use JSX, we are configuring the ability to use it later.

Next, modify the "dev" script in package.json to match the following:

```
"dev": "bun run --watch src/server.tsx"
```

The --watch flag causes the Bun server to be restarted if any of the source files it is uses are modified. This does not include client-side files in the public directory.

Now replace the contents of src/server.tsx with the following. Each code snippet in the book is labelled by a directory name that corresponds to the current chapter name, and a file name.

JumpingIn/server.tsx

```
Line 1 import {type Context, Hono} from 'hono';
     - import {serveStatic} from 'hono/bun';
     -
     - const app = new Hono();
     5
     - // Serve static files from the public directory.
```

```
-  app.use('/*', serveStatic({root: './public'}));
-
-
10 app.get('/version', (c: Context) => {
-    // Return a Response whose body contains
-    // the version of Bun running on the server.
-    return c.text(Bun.version);
-  });
15
-  export default app;
```

Context is a class defined by the Hono framework. An instance of this class is passed to all the methods that define endpoints such as app.get. The Context parameter provides access to request headers, path parameters, query parameters, and the request body.

Servers for htmx applications play two roles. First, they serve static files such as HTML, CSS, JavaScript, and images. The server code above implements this on line 7. Second, they respond to certain HTTP requests, typically returning HTML or text. The server code above implements this on line 10.

We have completed our work on the server-side code and are ready to focus on the client-side. Start by creating the public directory at the root of the project. Then create the file index.html in the public directory with the following content.

**JumpingIn/index.html**

```
Line 1 <html>
-        <head>
-          <title>htmx Demo</title>
-          <link rel="stylesheet" href="styles.css" />
5          <script src="https://unpkg.com/htmx.org@1.9.11"></script>
-        </head>
-        <body>
-          <button hx-get="/version" hx-target="#version">Get Bun Version</button>
-          <div id="version"></div>
10       </body>
-      </html>
```

The hx-get attribute on line 8 specifies that when the button element is triggered (clicked) an HTTP GET request should be sent to the endpoint at /version.

The hx-target attribute specifies that the HTML returned by the endpoint should replace the innerHTML of the element with id "version" on line 9. This will only happen if the response code is between 200 and 299, which indicates success. The innerHTML of an element encompasses all the HTML it contains.

We are obtaining the htmx library from a CDN. Alternatively, it can be manually downloaded or installed with npm (or bun) so it can be served along with other static files.

Create the file styles.css in the public directory with the following content:

JumpingIn/styles.css

```
body {
  font-family: sans-serif;
}

button {
  border-radius: 0.5rem;
  margin-bottom: 1rem;
  padding: 0.5rem;
}
```

If the local server is still running, stop it by pressing ctrl-c. Then enter bun dev to restart it.

Browse localhost:3000 again, and click the “Get Bun Version” button. Verify that a version number is displayed below the button.

Get Bun Version

v1.1.3

There you have it—first project done!

Take a moment to consider how the same application could be implemented in other web frameworks you have used. What code would be required to send an HTTP request when a button is clicked, and to insert the response into the current page? What code would be required to implement the end-point?

In the future when you want to create a new project that uses Bun, Hono, and htmx, rather than repeating all the steps above, just copy this project and modify the code.