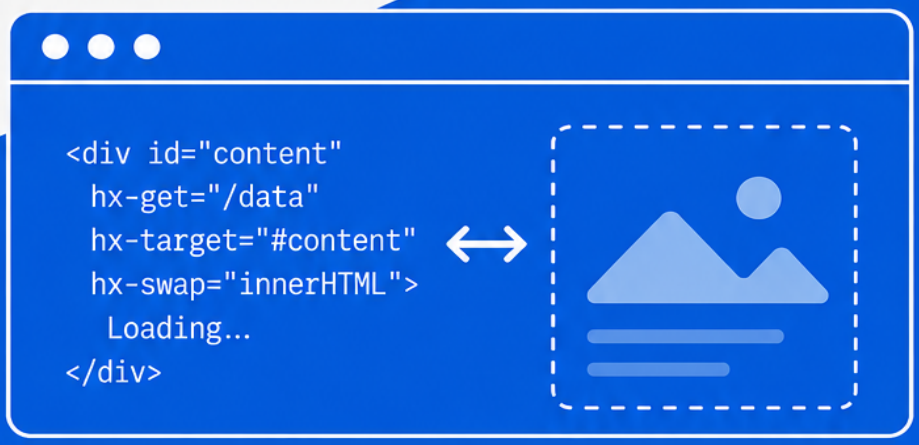
{ htmx }

HTMX IN PRACTICE

BUILDING DYNAMIC WEB INTERFACES WITH SERVER-DRIVEN FRONTEND DEVELOPMENT

4.0 INCLUDES HTMX 4.0 CONTENT



EXTEND HTML
Use the tools you already know.

LESS JAVASCRIPT
Build rich interfaces with less client code.

PRODUCTION READY
Patterns and techniques that scale.

EITAN SHALOM

HTMX in Practice

Building Dynamic Web Interfaces with Server-Driven Frontend Development

Steve Publications

This book is available at https://leanpub.com/htmxinpractice

This version was published on 2026-07-26

Contents

Building Dynamic Web Interfaces with Server-Driven Frontend Development

About this book

HTMX has emerged as one of the most talked-about tools in modern web development, offering a radically different approach to building interactive interfaces. This book takes you from zero knowledge of HTMX through advanced production patterns, showing you how to build rich, dynamic web applications by extending HTML rather than replacing it with client-side JavaScript. Through real-world examples, detailed code walkthroughs, and clear explanations of the underlying principles, you will learn not just how to use HTMX, but why its approach works and when to reach for it in your own projects. Whether you are a backend developer who wants to add interactivity without learning a frontend framework, or an experienced frontend engineer curious about hypermedia-driven alternatives, this book provides a comprehensive, practical guide to the entire HTMX ecosystem.

Introduction: The Case for Server-Driven Frontends

The JavaScript Fatigue Problem

It is 2026, and the modern web developer faces a bewildering array of choices. Build a single-page application with React, Vue, or Svelte? Reach for a meta-framework like Next.js, Nuxt, or Remix? Use a component library, a UI kit, or build from scratch? Manage state with Redux, Zustand, Pinia, or the framework's built-in solution? Bundle with Webpack, Vite, or esbuild? Each decision branches into more decisions, and the cumulative weight of all these choices has produced what many developers call "JavaScript fatigue."

The numbers tell part of the story. The React core library alone is approximately 7.4 KB minified and 2.8 KB gzipped, while React combined with ReactDOM runs about 43 KB gzipped [1]. But a real application needs routing, state management, data fetching, form handling, and UI component libraries. A typical production React application ships with a JavaScript bundle of 200 to 500 kilobytes minified and gzipped, and that number can grow well beyond a megabyte for feature-rich applications. On a low-end Android phone over a 3G connection, downloading, parsing, and executing that much JavaScript can take several seconds before the user sees anything interactive.

This is not to say that client-side frameworks are wrong or bad. They solve real problems: complex state management, rich animations, deep component composition, and offline capability. But they also introduce complexity that many projects do not need. The average content management dashboard, the typical CRUD application, the standard e-commerce product page: these do not require a virtual DOM, a build pipeline, or a separate frontend repository.

What HTMX Actually Is

HTMX is a small JavaScript library, approximately 14 KB when minified and gzipped [2]. For comparison, React combined with ReactDOM is about 43

KB gzipped, and a typical production React application with routing, state management, and UI libraries easily reaches 200 to 500 KB gzipped or more [5]. HTMX does not replace HTML with a component model. It does not introduce a new templating language or require a build step. Instead, it extends HTML with custom attributes that enable AJAX requests, CSS transitions, WebSockets, and Server-Sent Events directly in your markup.

Consider a simple example. In a traditional React application, building a button that fetches data and updates part of the page requires defining a component, managing state with hooks, handling loading states, error states, and the actual UI rendering. In HTMX, the same interaction looks like this:

```
<button hx-get="/api/items" hx-target="#results" hx-swap="innerHTML">
  Load Items
</button>
<div id="results"></div>
```

That is it. When the user clicks the button, HTMX issues a GET request to `/api/items`, takes the HTML response, and replaces the contents of the `#results` div. No JavaScript component, no state management, no build step. The server returns ready-to-render HTML, and HTMX handles the rest.

HTMX was created by Carson Gross, a professor of software engineering at Montana State University and co-author of the book *Hypermedia Systems*. It is the successor to intercooler.js, an earlier experiment in extending HTML with AJAX capabilities. HTMX has grown into a mature library with an active community, a rich extension ecosystem, and adoption across a wide range of projects from personal side-projects to enterprise applications.

Who Should Read This Book

This book is for web developers at any level who want to understand HTMX and use it effectively in production. The ideal reader has some familiarity with HTML, CSS, and basic server-side programming, but you do not need experience with any particular backend framework or frontend library. If you have built even a simple website that displays data from a database, you have the background to benefit from this book.

Backend developers who find themselves uncomfortable writing complex client-side JavaScript will appreciate HTMX's approach of keeping application

logic on the server. Frontend engineers who are curious about alternatives to the SPA model will find a rigorous exploration of the trade-offs and capabilities of hypermedia-driven development. Full-stack developers will discover a way to unify their mental model across the entire stack, reducing the friction between frontend and backend concerns.

How to Use This Book

The book is organized to build your understanding progressively. The first few chapters establish the philosophical and technical foundations of HTMX, explaining why its approach works and how it differs from client-side frameworks. You will learn every core attribute, understand the request lifecycle, and see how servers respond to HTMX requests.

The middle chapters dive into practical patterns: forms and validation, state management, authentication, error handling, accessibility, and performance optimization. Each chapter provides complete, production-ready code examples that you can adapt for your own projects. You will learn not just what works, but why certain patterns are preferred and what pitfalls to avoid.

The final chapters cover the broader ecosystem: extensions, backend framework integration, testing and debugging, application architecture, and deployment. By the end, you will be able to evaluate whether HTMX is the right tool for your project, build robust applications with it, and understand its limitations and boundaries.

Throughout the book, you will encounter comparisons with traditional JavaScript frameworks like React, Vue, and Svelte. These comparisons are not meant to dismiss those tools but to provide context for when HTMX makes sense and when a client-side framework might still be the better choice. The goal is to give you enough understanding to make informed decisions about your technology stack, rather than advocating for any single approach.

The code examples in this book use a variety of backend technologies, including Python (Django, Flask, FastAPI), Ruby on Rails, PHP/Laravel, and Go. Each example focuses on the HTMX patterns rather than framework-specific details, so you can follow along regardless of your preferred backend. Server-side code is shown where it illuminates the HTMX pattern, but the emphasis is always on the HTML attributes and the client-server interaction model.

The TaskFlow Running Project

Throughout this book, you will follow a single running project called **TaskFlow**, a team task management dashboard. TaskFlow starts as a simple list of tasks in Chapter 1 and grows progressively more sophisticated as each chapter introduces new concepts. By the end of the book, TaskFlow will include authentication, real-time notifications, file attachments, inline validation, error handling, and a fully tested production-ready codebase.

The project is designed so that you can see how earlier patterns compound into a complete application. When Chapter 5 covers forms, you will add task creation to TaskFlow. When Chapter 8 covers authentication, you will protect TaskFlow routes and add user profiles. When Chapter 14 covers testing, you will write tests for the TaskFlow endpoints you have built throughout the book.

You can follow along with TaskFlow using any backend framework, but the primary examples use Python with Flask for brevity. The HTMX patterns are identical regardless of your backend choice. All code for the TaskFlow project is available in a single repository that you can clone and run locally.

Chapter 1: Foundations – HTML as the Interface

HTMX rests on a simple but profound idea: HTML already has everything you need to build dynamic web interfaces. The browser is a sophisticated hypermedia engine, and HTTP is a mature protocol for exchanging structured data. What if, instead of building an entirely new system on top of these foundations, we simply extended them with the capabilities they are missing?

This chapter explores the philosophical and technical foundations that make HTMX possible. We will look at how web development evolved from static pages to single-page applications, why the pendulum is swinging back toward server-driven rendering, and what principles guide HTMX's design decisions. By the end of this chapter, you will understand not just how HTMX works, but why it works the way it does.

The Evolution from Static Pages to SPAs

The earliest websites were collections of static HTML pages. Each page was a complete document, linked together with anchor tags. When a user clicked a link, the browser made a full HTTP request, the server returned a new HTML document, and the browser replaced the entire view. This model was simple, predictable, and worked everywhere.

In the early 2000s, AJAX (Asynchronous JavaScript and XML) changed this picture. Developers could make background HTTP requests without reloading the page, fetch data from the server, and update parts of the DOM with JavaScript. Gmail, launched in 2004, demonstrated what was possible: a web application that felt as responsive as a desktop program.

Over the next decade, this approach evolved into the single-page application (SPA) model. Libraries like jQuery simplified AJAX calls, then frameworks like Angular, React, and Vue provided structured ways to manage the growing complexity of client-side state, routing, and rendering. The SPA became the dominant paradigm for web development, with entire applications running as

JavaScript programs that fetched JSON data from APIs and rendered HTML on the client.

The SPA model brought genuine benefits. Applications could feel incredibly responsive, with smooth transitions between views and instant user feedback. Complex interactive features like drag-and-drop, real-time collaboration, and rich text editing became feasible in the browser. The separation of frontend and backend allowed teams to scale independently, with dedicated frontend engineers optimizing the user interface while backend engineers focused on data and business logic.

But the SPA model also introduced significant costs. JavaScript bundles grew from tens of kilobytes to megabytes. Build pipelines required Node.js, bundlers, transpilers, and hot-module reloading. State management became its own discipline, with libraries like Redux adding hundreds of lines of boilerplate for simple data flows. The gap between frontend and backend created friction: duplicated validation logic, inconsistent error handling, and the perpetual challenge of keeping client-side state synchronized with server-side truth.

Most importantly, the SPA model often delivered poor initial load performance. Before a user could interact with the application, the browser had to download, parse, compile, and execute a large JavaScript bundle. On slow connections or low-end devices, this delay was noticeable and frustrating. Server-side rendering (SSR) frameworks like Next.js attempted to address this by pre-rendering HTML on the server, but they added yet another layer of complexity.

Progressive Enhancement Revisited

Progressive enhancement is a design philosophy that has existed since the early days of the web. The core principle is simple: build a baseline experience that works for everyone, then add advanced features for users with more capable browsers. A form should submit normally if JavaScript is disabled, but can be enhanced with AJAX submission and inline validation when JavaScript is available.

The SPA model largely abandoned progressive enhancement. Most SPAs deliver a blank screen without JavaScript, requiring the entire application to function client-side. This works well in controlled environments where you

can assume modern browsers and reliable connections, but it fails for users on older devices, in areas with poor connectivity, or with accessibility tools that behave differently.

HTMX brings progressive enhancement back into mainstream web development, but with a twist. Rather than building a fully functional application without JavaScript and then adding enhancements, HTMX makes JavaScript essential to the interactive experience while keeping the HTML itself meaningful and self-contained. The `hx-boost` attribute, for example, converts all links and forms within an element into AJAX requests. If JavaScript is disabled, those links and forms still work as normal navigation. The application degrades gracefully rather than failing completely.

This approach has practical implications for real-world applications. Consider a content management system used by journalists in the field. They may be working on older hardware, on unreliable connections, or with browsers that have limited JavaScript support. An HTMX-based interface will continue to function, perhaps more slowly, but without the catastrophic failure of a blank screen.

The Hypertext Philosophy

At its core, HTMX is an implementation of the hypertext philosophy. The term "hypertext" was coined by Ted Nelson in 1963 to describe text that contains links to other text, creating a non-linear structure for information. The World Wide Web, invented by Tim Berners-Lee in 1989, is the most successful implementation of hypertext ever created.

The web's original design is a complete application framework. HTTP provides the communication protocol, HTML provides the presentation and navigation model, and URLs provide universal addressing. Roy Fielding, the architect of REST, described this architecture in his doctoral dissertation as a "hypermedia-driven application" where the client and server communicate through representations that contain links to possible next states.

Modern web development has largely moved away from this vision. JSON APIs return data without links, forcing the client to know about available actions. Client-side frameworks maintain their own routing tables, disconnected from the server's understanding of the application state. The result is a system

where the client and server must be developed and deployed in lockstep, with shared schemas and coordinated versioning.

HTMX returns to the hypertext model. When a user clicks an HTMX-enhanced link, the server returns HTML that contains the next view along with links to subsequent actions. The client does not need to know about available endpoints or construct URLs programmatically. It simply renders what the server sends and lets the user navigate through hypermedia controls.

This approach has profound implications for maintainability and evolution. New features can be added by creating new server endpoints and linking to them from existing pages, without modifying the client code. The application evolves organically, like the web itself, rather than requiring coordinated releases of frontend and backend components.

Why Less JavaScript Can Mean More

The decision to use less JavaScript is not a rejection of the language itself. JavaScript is a powerful, flexible language that runs everywhere. But using it for everything (routing, state management, rendering, data fetching, form handling, animations, and more) creates applications that are difficult to understand, test, and maintain.

HTMX takes a different approach: use HTML for what HTML is good at (structure, semantics, navigation), use CSS for what CSS is good at (presentation, transitions), use the server for what the server is good at (data access, business logic, security), and use JavaScript only where it adds genuine value.

The result is code that is easier to read and modify. An HTMX interaction is declared inline in the HTML, right next to the element it affects. You do not need to navigate between a template file, a component file, a state management file, and an API service file to understand what happens when a user clicks a button. Everything is visible in one place.

This principle, called "locality of behavior," is central to HTMX's design philosophy. When the code that defines behavior is physically close to the HTML element it affects, developers can understand and modify the application more quickly. This reduces cognitive load, speeds up development, and makes code reviews more effective.

Consider a comparison. In a React application, implementing a button that toggles a dropdown menu requires:

```
function Dropdown() {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <>
      <button onClick={() => setIsOpen(!isOpen)}>Menu</button>
      {isOpen && (
        <div className="dropdown">
          <a href="/item1">Item 1</a>
          <a href="/item2">Item 2</a>
        </div>
      )}
    </>
  );
}
```

The same interaction in HTMX with minimal JavaScript (or using the Alpine.js companion library) keeps the behavior local to the markup:

```
<button hx-get="/menu-content" hx-target="#dropdown" hx-toggle="show">
  Menu
</button>
<div id="dropdown"></div>
```

The HTMX version declares what happens when the button is clicked, where the response goes, and how it is displayed, all in the HTML itself. No component definition, no state hooks, no conditional rendering logic. The server returns the menu content as an HTML fragment, and HTMX handles the rest.

Setting Up Your First HTMX Project

Getting started with HTMX requires almost no setup. Unlike React or Vue, there is no build pipeline, no package manager dependency tree, and no configuration file to create. You can add HTMX to any existing web page by including a single script tag.

HTMX Version Compatibility

This book covers **HTMX v2.x and v4.x**. HTMX 2.0 was released in June 2024 as a stable, production-ready version. HTMX 4.0, currently in beta (v4.0.0-beta6 as of July 2026), introduces significant new capabilities while maintaining backward compatibility through a migration extension [57].

HTMX creator Carson Gross announced v4.0 in November 2025, skipping version 3 entirely. The release, dubbed "The Fetch()ening," moves HTMX from XMLHttpRequest to the modern fetch() API and adds streaming responses, built-in DOM morphing, and new markup elements [56]. The core philosophy remains unchanged: extend HTML with attributes to build dynamic interfaces without client-side frameworks.

Feature	**HTMX 1.x**	**HTMX 2.x**	**HTMX 4.x (beta)**
HTTP transport	XMLHttpRequest	XMLHttpRequest	fetch() API
CDN package name	`htmx.org`	`htmx.org`	`htmx.org@4.0.0-beta6`
Extensions location	Bundled in `/dist/ext/`	Separate npm packages (`htmx-ext-*`)	Same as 2.x
`hx-on` syntax	`hx-on:event="..."`	`hx-on::event="..."` (double colon)	`hx-on:` (single colon standard)
SSE/WebSocket	`hx-sse`, `hx-ws` attrs	Extension-only	Extension-based, now using fetch()
Attribute inheritance	Implicit (CSS-style)	Implicit	Explicit (`:inherited` modifier required)
Error responses (4xx/5xx)	No swap by default	No swap by default	Swap by default (configurable)
Streaming responses	Not supported	Not supported	Built-in via ReadableStream
DOM morphing	Extension only	Extension only (idiomorph)	Built-in (`innerMorph`, `outerMorph`)
Event naming	`htmx:beforeRequest`	`htmx:beforeRequest`	`htmx:before:request` (colon-separated)

Feature	HTMX 1.x	HTMX 2.x	HTMX 4.x (beta)
History caching	sessionStorage snapshots	sessionStorage snapshots	Network re-fetch on back navigation
Default timeout	0 (no timeout)	0 (no timeout)	60 seconds
IE support	Supported	Dropped	Not supported

Which version should you use?

- **HTMX 2.x**: Stable, production-ready, widely adopted. The safest choice for new projects today. All core attributes work identically in both versions.
- **HTMX 4.x**: Beta status with a planned stable release in early 2027. Includes powerful new features like streaming and morphing. Use if you want to experiment or need the new capabilities, but be prepared for potential changes.

This book teaches patterns that work across both versions where possible, with callouts for version-specific behavior. Code examples default to HTMX 2.x syntax for broad compatibility, with HTMX 4 additions shown alongside.

If you are migrating from HTMX 1.x to 2.x, refer to the official migration guide [3]. For 2.x to 4.x migration, see the HTMX 4 migration documentation [57].

Via CDN (quickest start)

Add this line to the `<head>` of your HTML document:

```
<!-- HTMX 2.x (stable, recommended for production) -->
<script src="https://cdn.jsdelivr.net/npm/htmx.org@2.0.10/dist/htmx.min.js"></
↪  /script>
```

Or for HTMX 4.x (beta):

```
<!-- HTMX 4.x (beta, new features) -->
<script src="https://cdn.jsdelivr.net/npm/htmx.org@4.0.0-beta6"></script>
```

That is all you need. HTMX is now loaded and ready to process any `hx-*` attributes on your page.

Via npm (for projects with a build step)

If your project already uses a package manager, you can install HTMX as a dependency:

```
# HTMX 2.x (stable)
npm install htmx.org@2.0.10

# HTMX 4.x (beta)
npm install htmx.org@4.0.0-beta6
```

Then import it in your JavaScript entry point or include it via your bundler's asset pipeline.

Via direct download (for offline or self-hosted deployments)

Download the minified file from the HTMX website and host it on your own server:

```
<script src="/static/htmx.min.js"></script>
```

Loading extensions

HTMX extensions are distributed as separate packages. Load them after the core library:

```
<!-- HTMX 2.x with extensions -->
<script src="https://cdn.jsdelivr.net/npm/htmx.org@2.0.10/dist/htmx.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/htmx-ext-sse@2.2.4/dist/sse.js"></script>
<script
  src="https://cdn.jsdelivr.net/npm/htmx-ext-ws@2.0.4/dist/ws.js"></script>

<!-- HTMX 4.x with extensions -->
<script src="https://cdn.jsdelivr.net/npm/htmx.org@4.0.0-beta6"></script>
<script src="https://cdn.jsdelivr.net/npm/htmx-ext-sse@4.0.0-beta6/dist/hx-sse.js"></script>
<script
  src="https://cdn.jsdelivr.net/npm/htmx-ext-ws@4.0.0-beta6/dist/hx-ws.js"></script>
```

In HTMX 2.x, extensions are enabled on specific elements using the `hx-ext` attribute. In HTMX 4.x, extensions are loaded via script tags and use event-based hooks rather than the `hx-ext` attribute. Details are covered in Chapter 12.

Your first HTMX interaction: the TaskFlow dashboard

Rather than a disconnected example, let us start building the TaskFlow project that will evolve throughout this book. Create a simple HTML file to verify that HTMX is working:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>TaskFlow - Team Task Dashboard</title>
  <script src="https://cdn.jsdelivr.net/npm/htmx.org@2.0.10/dist/htmx.min.js"
  ></script>
  <style>
    body { font-family: system-ui, sans-serif; max-width: 800px; margin: 2rem
    auto; padding: 0 1rem; }
    .task-list { list-style: none; padding: 0; }
    .task-item { padding: 0.75rem; border: 1px solid #ddd; border-radius: 4px;
    margin-bottom: 0.5rem; display: flex; justify-content: space-between;
    align-items: center; }
    .task-item.completed { opacity: 0.6; text-decoration: line-through; }
    button { padding: 0.25rem 0.75rem; border: 1px solid #ccc; border-radius:
    4px; cursor: pointer; background: white; }
    button:hover { background: #f5f5f5; }
  </style>
</head>
```

```
<body>
  <h1>TaskFlow</h1>

  <button hx-get="/api/tasks" hx-target="#task-list" hx-swap="innerHTML">
    Load Tasks
  </button>

  <ul id="task-list" class="task-list">
    <li>Click "Load Tasks" to fetch your tasks.</li>
  </ul>
</body>
</html>
```

On the server side, you need an endpoint that returns HTML fragments. Here is how it looks in several popular frameworks:

Python (Flask)

```
from flask import Flask

app = Flask(__name__)

# In production, this data would come from a database
TASKS = [
    {"id": 1, "title": "Set up project repository", "completed": True},
    {"id": 2, "title": "Design database schema", "completed": False},
    {"id": 3, "title": "Implement authentication", "completed": False},
]

@app.route('/api/tasks')
def get_tasks():
    html = ""
    for task in TASKS:
        cls = "task-item completed" if task["completed"] else "task-item"
        html += f'''<li class="{cls}" id="task-{task["id"]}">
          <span>{task["title"]}</span>
          <button hx-post="/api/tasks/{task["id"]}/toggle"
                  hx-target="#task-{task["id"]}"
                  hx-swap="outerHTML">
            Toggle
          </button>
        </li>'''
    return html
```

Ruby on Rails

```ruby
# app/controllers/api/tasks_controller.rb
class Api::TasksController < ApplicationController
  def index
    tasks = Task.all
    render html: tasks.map do |task|
      cls = task.completed ? "task-item completed" : "task-item"
      %(<li class="#{cls}" id="task-#{task.id}">
          <span>#{task.title}</span>
          <button hx-post="/api/tasks/#{task.id}/toggle"
                  hx-target="#task-#{task.id}"
                  hx-swap="outerHTML">
            Toggle
          </button>
        </li>).html_safe
    end.join("")
  end
end
```

PHP (Laravel)

```php
// routes/web.php
use App\Models\Task;

Route::get('/api/tasks', function () {
    $tasks = Task::all();
    $html = '';
    foreach ($tasks as $task) {
        $cls = $task->completed ? 'task-item completed' : 'task-item';
        $html .= "<li class=\"$cls\" id=\"task-{$task->id}\">
          <span>{$task->title}</span>
          <button hx-post=\"/api/tasks/{$task->id}/toggle\"
                  hx-target=\"#task-{$task->id}\"
                  hx-swap=\"outerHTML\">
            Toggle
          </button>
        </li>";
    }
    return $html;
});
```

When you click “Load Tasks,” HTMX sends a GET request to `/api/tasks`, receives the HTML fragment from the server, and replaces the contents of `#task-list` with the response. No page reload, no JavaScript component, no build step. The returned HTML includes its own HTMX attributes (the Toggle buttons), which HTMX automatically processes when inserted into the DOM.

Verifying HTMX is loaded

Open your browser's developer console and type `htmx`. If HTMX is loaded correctly, you will see the HTMX object with its methods and configuration options. If you see `undefined`, check that the script tag is correct and that there are no network errors loading the file.

This minimal setup demonstrates everything that makes HTMX compelling: the behavior is declared in HTML, the server returns ready-to-render content, and the browser handles the update automatically. Every feature we explore in this book builds on these same principles, and the TaskFlow project will grow with each chapter.

Chapter 2: Core Attributes and the HTMX API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

hx-get, hx-post, and HTTP Verb Attributes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

hx-target and Where Responses Land

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

hx-trigger – The Event System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

hx-swap and Content Replacement Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

hx-select for Surgical Extraction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

hx-push-url, hx-boost, and Navigation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

HTMX 4.x: New Attributes and Behaviors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 3: The Request Lifecycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

From Trigger to Response – Step by Step

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Out-of-Band Swaps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The hx-indicator Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Headers and Request Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

History Management with hx-push-url and hx-history

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 4: Server-Side Rendering Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Returning HTML Fragments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Full Page vs Partial Responses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Templating with HTMX in Mind

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The hx-on Server-Side Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Streaming and Incremental Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 5: Forms, Validation, and User Input

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The Simplest Possible Form

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Inline Validation Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Multi-Step and Wizard Forms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

File Uploads with hx-encoding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Form State and Preservation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 6: State Management Without a Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The DOM as Your State Container

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Server-Side State Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Hidden Inputs and hx-include

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

URL as State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

When to Use Client-Side State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 7: Events and Inter-Component Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

HTMX Built-In Events

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Custom Events with hx-on

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The htmx:configRequest Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Event Propagation and Scope

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Writing Minimal Glue Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 8: Authentication and Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Session-Based Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Login and Registration Flows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

CSRF Protection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Authorization and Conditional Rendering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Handling Expired Sessions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 9: Error Handling and Resilience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

HTTP Status Codes and HTMX Responses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The hx-error Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

User-Facing Error Messages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Retry Logic and Resilience

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Debugging HTMX Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 10: Accessibility and Progressive Enhancement

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Why HTMX Is Naturally Accessible

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

ARIA Live Regions and hx-swap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Keyboard Navigation Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Graceful Degradation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Testing Accessibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 11: Performance Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Measuring HTMX Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Caching Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Lazy Loading with hx-trigger

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Real-Time with WebSockets and SSE

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Request Coalescing and Throttling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 12: Extensions and the HTMX Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The Extension API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Must-Know Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Writing Custom Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Hyperscript as a Companion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The Broader Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 13: Integration with Backend Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Python: Django and Flask Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Python: FastAPI with HTMX

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Ruby on Rails Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

PHP and Laravel

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Go and Other Languages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 14: Testing and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Unit Testing Server Responses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Integration Testing with Playwright

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

End-to-End Testing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Browser DevTools for HTMX

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Common Debugging Scenarios

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Chapter 15: Application Architecture and Production Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Project Structure and Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The Monolith vs Microservices Question

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Deployment Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Monitoring and Observability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

When HTMX Is Not the Right Choice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Conclusion: The Future of Server-Driven Frontends

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

What We Have Built

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

The Bigger Picture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

Making the Right Choice for Your Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/htmxinpractice.