Over 35 Years Of Technology Training

Document Generated: 04/06/2026

Learning Style: Virtual Classroom

Technology:

Difficulty: Intermediate

Course Duration: 3 Days

Next Course Date: June 1, 2026

Server-Driven Web Apps with htmx: the HTML-First Approach (TT4005)

About This Course:

Build modern, interactive web applications without writing JavaScript. This course teaches you to create dynamic, server-driven interfaces using htmx, a lightweight

library that extends HTML with powerful attributes for AJAX, WebSockets, and real-time updates. You'll learn to build responsive UIs that rival single-page applications, but with dramatically simpler architecture and better SEO.

Master progressive enhancement, hypermedia-driven design, and server-side rendering patterns. Create interactivity such as auto-updating dashboards, infinite scroll feeds, modal dialogs, form validation, and live search without complex JavaScript frameworks. You'll work with Python/Flask backends to build applications that are faster to develop, easier to maintain, and more accessible than React or Vue equivalents.

Find out when htmx excels (content-heavy sites, admin dashboards, CRUD applications) and when traditional SPAs might be better. Build real-world projects.

Course Objectives:

- Evaluate use cases where htmx provides advantages over JavaScript frameworks, including faster development cycles, improved SEO, reduced complexity, and better accessibility
- Implement core htmx functionality using attributes like hx-get, hx-post, hx-target, hx-swap, and hx-trigger to create interactive web interfaces
- Build dynamic forms with real-time validation, loading indicators, error handling, and server-side integration
- Create server endpoints in Python/Flask that return HTML fragments for partial page updates
- Implement real-time features using polling, server-sent events, and trigger-based updates for live dashboards and notifications
- Build single-page application experiences using hx-boost, hx-push-url, and browser history management
- Manage application state and data flow between client and server using hx-vals, hx-params, and htmx event hooks
- Deploy production-ready htmx applications with optimized performance, accessibility, and SEO
- Design architectures that maximize the benefits of hypermedia-driven development while understanding its trade-offs

Audience:

- Backend developers who want to add modern interactivity without learning heavy JavaScript frameworks

- Full-stack developers seeking simpler alternatives to React/Vue/Angular for dynamic UIs
- Python/Django/Flask developers building interactive admin interfaces and dashboards
- Frontend developers interested in server-driven architecture and progressive enhancement
- Teams looking to reduce frontend complexity, maintenance burden, and build times
- Developers building content-heavy sites where SEO and accessibility are critical priorities

Prerequisites:

- Students should have solid HTML and CSS skills. Understanding of HTTP fundamental concepts (requests, responses, status codes) and familiarity with basic web development concepts like forms, the DOM, and client-server architecture is helpful.

Course Outline:

1) What htmx is and Setup

Understand the philosophy behind htmx and get your development environment ready. Learn why hypermedia-driven applications are making a comeback and when htmx is the right choice for your project.

- The origins of htmx and Intercooler.js
- REST and HATEOAS principles
- The role of hypermedia in frontend architecture
- Comparison to JavaScript-heavy approaches
- Use cases where htmx excels
- Project structure and HTML setup
- CDN vs npm installation
- Setting up a local server and dev tools
- Testing the htmx script load

Set up an HTML page with htmx loaded via CDN, create a simple Flask endpoint.

2) Core htmx Attributes & Patterns

Master the fundamental building blocks of htmx—the attributes that transform static

HTML into interactive applications without writing JavaScript.

- hx-get, hx-post, hx-put, hx-delete
- hx-target, hx-trigger, and hx-include
- hx-swap and swap modes (innerHTML, outerHTML, afterbegin, etc.)
- Creating backend endpoints
- HTTP method overrides and safety
- Working with hidden inputs and CSRF tokens
- Real-time updates with polling and triggers
- Inspecting requests and responses with dev tools
- Debugging common issues
- Accessibility considerations

Build an interactive todo list with add, delete, and mark-complete functionality.

3) Backend Integration

Learn to create server endpoints that work seamlessly with htmx, with deep focus on Python/Flask patterns that transfer to other backends.

- Python: Flask (light endpoints), Django (forms/templates), CSRF

Build a complete contact form system with Flask backend.

4) Forms and Validation

Master form handling in htmx—from simple submissions to complex multi-step workflows with real-time validation and elegant user feedback.

- Form event lifecycle in htmx
- hx-indicator and spinners
- Error messaging and status elements
- HTML5 validation and custom server responses
- Integrating with backend form validators

Create a multi-field user registration form with real-time validation.

5) DOM Manipulation & Conditional Rendering

Control what users see and when they see it, using server logic to drive UI changes dynamically without client-side JavaScript.

- Inline conditionals using server logic
- DOM fragment targeting
- Lazy loading with hx-trigger on scroll
- Content refresh without user input

Build an infinite-scroll news feed that lazy-loads articles as users scroll down.

6) State and Data Flow

Manage application state and coordinate data between client and server in the htmx paradigm where the server is the source of truth.

- hx-vals, hx-params, and passing state
- Simulating data binding with dynamic values
- Triggering updates on input change, focus, blur, or custom events
- htmx event hooks: htmx:beforeSend, etc.
- Context-aware updates
- Keeping UI in sync with backend data changes
- Linking components without JavaScript routing

Build a shopping cart system.

7) Performance and Testing

Optimize htmx applications for speed and reliability, and implement comprehensive testing strategies to ensure quality.

- Optimize htmx applications for speed and reliability
- Implement comprehensive testing strategies to ensure quality.
- Lazy loading content and images
- Server-side performance tips
- Testing HTML responses
- Debugging with dev tools and htmx events

Take the todo app and optimize it. Write backend unit tests for all endpoints.

8) Building SPAs with htmx

Create single-page application experiences that provide smooth navigation and instant updates without the complexity of traditional SPA frameworks.

- The importance of an SPA
- Page transitions with hx-boost
- Using hx-push-url for history tracking
- Partial loading patterns
- SPA vs MPA tradeoffs
- Linking views together with state

Convert a traditional multi-page blog application into an SPA experience.

9) Deployment and Best Practices

Take your htmx application to production with confidence, following industry best practices for performance, security, and maintainability.

- Hosting options (Vercel, Netlify, Heroku)
- SEO optimization with server-rendered pages
- Accessibility and semantic HTML
- Performance audits
- Bundling and minification tips

Deploy your htmx application to a hosting platform of choice (Heroku or Railway for example).

10) Optional: End-to-End Testing

Implement comprehensive end-to-end testing for htmx applications using Playwright to ensure reliability and catch bugs before production.

- Using Playwright for E2E Testing with htmx

Add Playwright tests to the todo app that verify: adding a todo, deleting a todo, marking complete, and form validation.

11) Optional: UI/UX and Animations

Polish your htmx applications with smooth transitions and professional feedback patterns that make server-driven UIs feel responsive and modern.

- Using CSS transitions with hx-swap and partial updates
- Leveraging hx-indicator for loading animations
- Animating content appearance with opacity, scale, and slide-in effects
- Delaying or staging swaps for animation timing
- Integrating with animation libraries like Animate.css
- Enhancing user feedback with class toggles and conditional states
- Best practices for accessible and non-distracting animations

Enhance the todo app from earlier with animations

12) Optional: Custom Extensions and Alpine.js Integration

Extend htmx's capabilities for edge cases and add lightweight client-side logic when server-driven patterns aren't enough.

- htmx.config and customizing behavior
- Registering an extension
- Overriding request/response handling
- Using Alpine.js for client-side state
- Lightweight two-way interaction

- Integrating third-party JavaScript modules

Create a modal dialog system using Alpine.js for open/close state management.