HTMX

webdev : et si on simplifiait ? Une autre route que les gros frameworks JS

Pierre Gambarotto

INSTITUT DE MATHÉMATIQUES
de TOULOUSE

ÉCRIRE UNE APPLICATION WEB — PETITE HISTOIRE

Le web originel (1991–2005)

- Deux verbes HTTP : **GET** (lire) et **POST** (envoyer)
- Deux éléments interactifs dans le navigateur : **liens** (<a>) et **formulaires** (<form>)
- Le serveur génère la page entière, le navigateur l'affiche
- Ça marche, c'est simple… mais chaque action recharge la page complète

L'ÈRE JAVASCRIPT ET LES SPA (2005–AUJOURD'HUI)

- **XMLHttpRequest** (2005) puis **fetch** : requêtes sans rechargement
- Problème résolu : la lenteur de recomposition complète des pages
- Mais nouvelle complexité : l'app tourne *dans* le navigateur
- Apparition de frameworks toujours plus gros : jQuery → Backbone → Angular → React → Next.js…

LE NAVIGATEUR DEVIENT UN *OS APPLICATIF*

- **Routeur** client (React Router, Vue Router…)
- **Store** d'état global (Redux, Vuex, Pinia, Zustand…)
- **Bundler** (Webpack, Vite, Rollup, esbuild, Turbopack…)
- **Transpileur** (Babel, TypeScript, SWC…)
- **SSR** — Server-Side Rendering (Next.js, Nuxt…)
- **SSG** — Static Site Generation
- **Hydration** — rattacher le JS au HTML pré-rendu
- **Tree-shaking**, **code-splitting**, *lazy loading*…
- **Hot Module Replacement** (HMR)
- **Virtual DOM**, **reconciliation**

Pour faire… ce que HTML + formulaires faisaient déjà.

“The actual design of the Web involved the creation of open standards – and getting people to agree to use them globally.” — Tim Berners-Lee

HTMX — UNE RÉPONSE TARDIVE MAIS FIDÈLE AUX STANDARDS

- Même constat de départ : le rechargement complet est trop lent
- Réponse différente : étendre HTML au lieu de le remplacer par JS
- Tout élément peut déclencher une requête (`hx-get`, ~hx-post~…)
- Le serveur renvoie du **HTML partiel**, le navigateur l'insère dans le DOM
- Résultat : la fluidité d'une SPA, la simplicité du web originel

INFLATION CÔTÉ FRONT (TAILLE & PERFS)

- 2015 → 2025 : page d'accueil mobile médiane 845 KB → 2 362 KB (+202,8%)
- 2025 : poids médian ≈ 2,6 MB (mobile) / 2,9 MB (desktop)
- JavaScript médian : 632 KB (mobile) / 697 KB (desktop)
- 22–23 requêtes JS en médiane par page
- À 2–3 MB : ~63% (desktop) et ~53% (mobile) passent les Core Web Vitals

Source : Web Almanac 2025, HTTP Archive

La simplicité est une grande vertu, mais elle exige un travail acharné pour être atteinte, ainsi qu'une éducation pour être appréciée. Et, pour aggraver les choses, la complexité se vend mieux. — Edsger Dijkstra

PHILOSOPHIE — RETOUR AUX FONDAMENTAUX DU WEB

LE WEB EST HYPERMÉDIA

- HTML est *le* format hypermédia du web
- HATEOAS : *Hypermedia As The Engine Of Application State*
- Le serveur envoie de l'HTML, pas du JSON + un moteur de rendu
- htmx étend les capacités hypermédia de HTML (tout élément peut faire une requête, tout événement peut la déclencher)

HTMX EN ACTION — EXEMPLES DE PATTERNS

Recherche temps réel

```
<input type="search" name="q"
       hx-get="/search" hx-trigger="keyup changed delay:300ms"
       hx-target="#results" />
<div id="results"></div>
```

INFINITE SCROLL

```
<tr hx-get="/contacts/?page=2"
    hx-trigger="revealed"
    hx-swap="afterend">
  <td>Chargement…</td>
</tr>
```

FORMULAIRES DYNAMIQUES

```
<select name="pays" hx-get="/regions" hx-target="#regions">
  <option value="FR">France</option>
</select>
<select id="regions" name="region"></select>
```

CHARGEMENT PARESSEUX

```
<div hx-get="/stats" hx-trigger="load" hx-swap="innerHTML">
  <img class="htmx-indicator" src="/spinner.svg"/>
</div>
```

démo

LOCALITY OF BEHAVIOUR (LOB)

> *The behaviour of a unit of code should be as obvious as possible by looking only at that unit of code. — Carson Gross*

- Le comportement est lisible *dans le HTML* :

```
<button hx-get="/contacts" hx-target="#list" hx-swap="innerHTML">
  Charger les contacts
</button>
```

- Pas besoin de chercher dans un fichier JS séparé

UN SEUL ÉTAT : LE SERVEUR

- SPA : double état (client + serveur) → synchronisation permanente
- htmx : un seul état côté serveur, rendu en HTML
- Pas de store Redux/Vuex/Pinia à maintenir
- Pas de sérialisation JSON / désérialisation / validation côté client

AVANTAGES CONCRETS VS SPA

Critère	SPA (React, Vue…)	htmx + serveur
État	Double (client + serveur)	Serveur uniquement
Build / toolchain	Webpack, Vite, Babel, TS…	Aucun (ou minimal)
Logique métier	Dupliquée front/back	Centralisée côté serveur
SEO	Nécessite SSR/SSG	Natif (HTML)
Courbe d'apprentissage	Framework + écosystème JS	HTML + attributs htmx
Taille équipe requise	Front + back séparés	Fullstack, équipe réduite
Langage backend	Contraint (Node ou API JSON)	Libre (Python, Go, Ruby…)
Déploiement	2 pipelines (front + API)	1 seul déploiement

CONTEXTE LABO / PETITES ÉQUIPES

- Équipes de 1 à peu de personnes pour le développement
- Maintenance sur 10+ ans (pérennité des outils)
- Rotation du personnel (mutations, départs)
- Pas de budget pour un·e dev front dédié·e

POURQUOI HTMX RÉPOND À CES CONTRAINTES

- **Pérennité** : HTML ne casse pas — les pages de 1995 fonctionnent encore
- **Maintenabilité** : on lit le HTML, on comprend le comportement
- **Compétences** : un·e admin sys qui connaît Python/Flask peut écrire l'app

POURQUOI HTMX (SUITE)

- **Sécurité** : surface d'attaque réduite (pas d'API JSON exposée)
- **Sobriété numérique** : moins de JS = moins de CPU côté client

LIMITES

htmx n'est **pas** la solution universelle :

- **Apps très interactives** : éditeur collaboratif, tableur, jeu temps réel
- **Offline-first** : htmx nécessite le réseau
- **État client complexe** : drag & drop, undo/redo multi-niveaux

LIMITES (SUITE)

- **Écosystème composants** : pas d'équivalent Material UI, Vuetify…
- **Équipes front existantes** : coût de migration si déjà sur React

QUAND CHOISIR QUOI ?

Besoin	Recommandation
CRUD, back-office, dashboards	✅ htmx
Site avec formulaires dynamiques	✅ htmx
App interne labo / petite équipe	✅ htmx
App collaborative temps réel (Google Docs-like)	❌ Framework JS (React…)
App offline-first / PWA lourde	❌ Framework JS
App avec état client très riche	⚠️ Hybride ou Framework
Prototype rapide / MVP	✅ htmx

Règle simple : si votre app peut fonctionner avec des pages et des formulaires, htmx suffit.

CE QUE LES SPA FONT BIEN : L'APPROCHE COMPOSANT

- Encapsulation : HTML + CSS + logique dans une unité réutilisable
- Composition : assembler des composants comme des briques
- Isolation : un composant ne « fuit » pas sur le reste de la page
- C'est l'apport majeur de React, Vue, Svelte — et ça manque côté htmx

MAIS… C’EST UN STANDARD DU WEB

Les **Web Components** existent nativement dans le navigateur :

- **Custom Elements** : `<user-card>`, `<search-widget>` — vos propres balises HTML
- **Shadow DOM** : encapsulation CSS et DOM, isolation réelle
- **HTML Templates** (`<template>`) : fragments HTML inertes, instanciables
- **Slots** : points d’insertion de contenu dans un composant

Sources : MDN Web Components

LE PROBLÈME : TOUT EST EN JAVASCRIPT

```
class UserCard extends HTMLElement {
  connectedCallback() {
    const shadow = this.attachShadow({ mode: 'open' });
    shadow.innerHTML = `
      <style>:host { display: block; border: 1px solid #ccc; }</style>
      <h3>${this.getAttribute('name')}</h3>
    `;
  }
}
customElements.define('user-card', UserCard);
```

- Le Shadow DOM est créé *en JS*, au runtime
- Pas de rendu tant que JS n'a pas tourné → mauvais pour SEO, perfs, accessibilité
- On retombe dans le même piège que les SPA…

DECLARATIVE SHADOW DOM (DSD) — LE STANDARD MANQUANT

Le DSD permet de déclarer un Shadow DOM directement en HTML, sans JavaScript.

```
<user-card>
  <template shadowrootmode="open">
    <style>:host { display: block; border: 1px solid #ccc; }</style>
    <h3><slot name="name">Anonyme</slot></h3>
  </template>
  <span slot="name">Alice Martin</span>
</user-card>
```

- Le navigateur crée le Shadow DOM au *parsing* du HTML
- Rendu immédiat, sans attendre JS
- SEO-friendly : le contenu est dans le HTML source
- Compatible avec le modèle htmx : le serveur peut renvoyer des fragments avec DSD

Sources : Chrome — Declarative Shadow DOM • Server-first Web

Components with DSD, htmx and Islands

DSD + HTMX : LE MEILLEUR DES DEUX MONDES

- **htmx** gère les interactions serveur (requêtes, fragments HTML)
- **DSD** gère l'encapsulation côté client (composants, style isolation)
- Pas de build, pas de framework JS
- Le serveur reste la source de vérité
- Composants réutilisables *et* server-rendered

Le web rattrape enfin les frameworks — avec des standards.