DevFest

Modena

2025

Hypermedia-Driven Applications: The Browser Reloaded!

Marco Breveglieri
Software Developer @ABLS Team

Google Developer Groups

{ DevFest }

Modena

Marco Breveglieri

- 👨‍💻 Software Developer
- 👨‍🏫 Trainer & Consultant
- 🎥 Tech Content Creator

Google Developer Groups

Homepage
https://www.breveglieri.it

Blog tecnico
https://www.compilaquindiva.com

Delphi Podcast
https://www.delphipodcast.com

Canale Twitch
https://twitch.tv/compilaquindiva

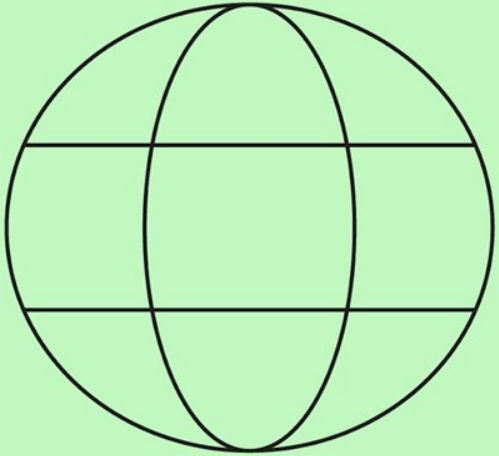
Canale YouTube
https://www.youtube.com/@compilaquindiva

Tutti gli altri social 👇
https://linktr.ee/marco.breveglieri

Google Developer Groups

{ DevFest }

Modena



C’era una volta il Web...

Come è cambiato il re 👑
dei linguaggi di markup,
ovvero l’**HTML**?

Google
Developer
Groups

30 anni di evoluzione: cosa è cambiato?

```
<form method="post" action="/submit">
  <input type="text" name="email">
  <input type="submit">Invia</input>
</form>
```

HTML 2.0 - 1995

```
<form method="post" action="/submit">
  <input type="text" name="email">
  <input type="submit">Invia</input>
</form>
```

HTML5 - 2025

Google Developer Groups

Eppur si muove...

HTML5
Audio/video
Semantica
Canvas/SVG
Form Validation
API JavaScript

HTML 2.0
Forms

1991 1995 1999 ... 2014

HTML 1.0
Testo e link

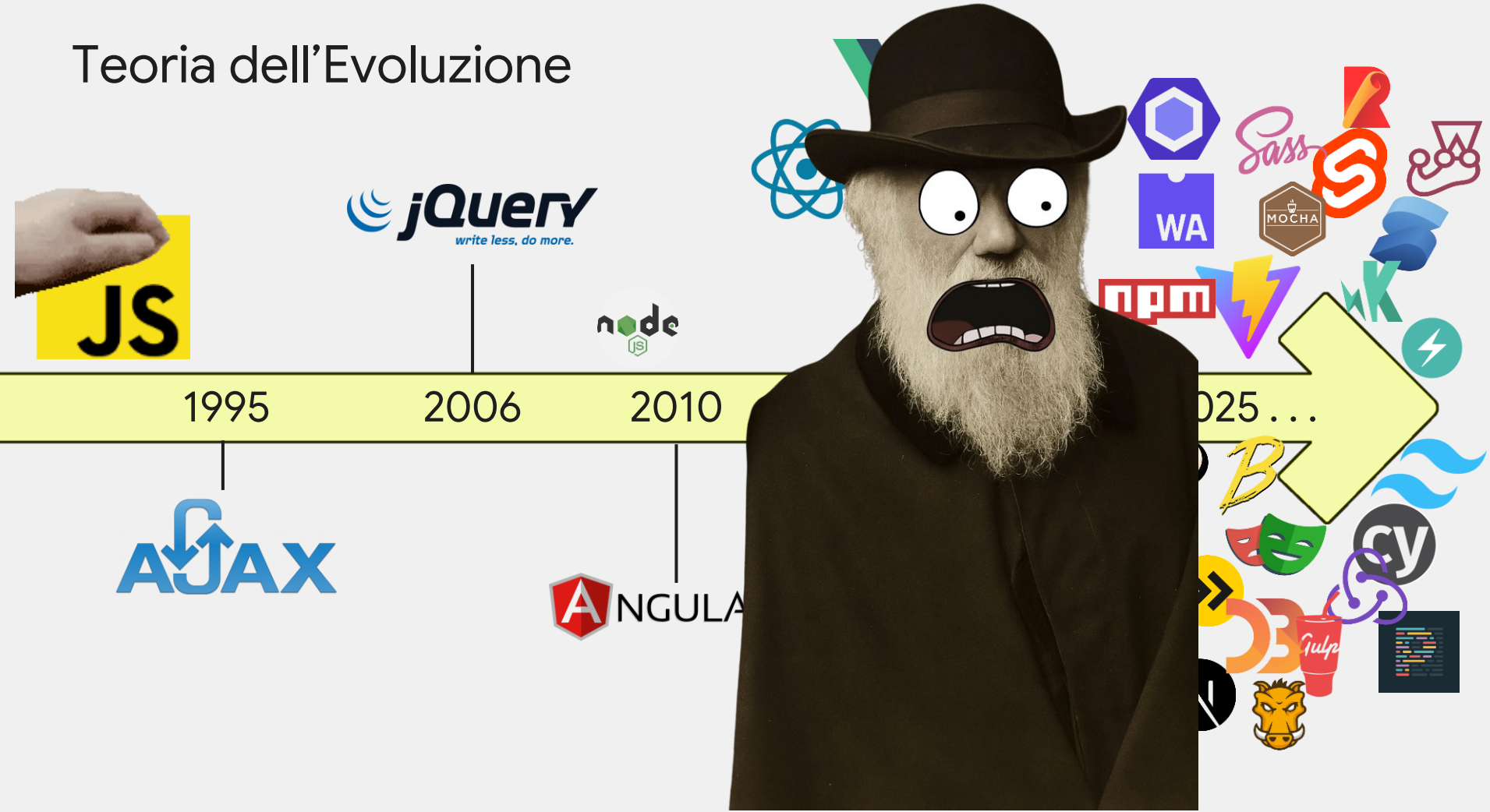
HTML 4.0
CSS
Embed
Frames

Il linguaggio HTML sa evolvere quando serve: perché è successo solo in alcuni aspetti?

HTTP, quanto ti amo! 💘

Comandi HTTP		Form
GET	DELETE	GET
POST	PATCH	POST
PUT	OPTIONS	

Teoria dell'Evoluzione



Per fare un semplice TODO, ci serve...

- ✓ 📦 Package Manager (npm, yarn, pnpm)
- ✓ 🔧 Bundler (Webpack, Vite, Rollup)
- ✓ 🔄 Transpiler (Babel, SWC)
- ✓ 📝 TypeScript (o JavaScript + linter)
- ✓ ✨ Formatter (Prettier, ESLint)

Google Developer Groups

Un framework e qualche libreria...

- ✓ ⚛️ Framework di scelta (React/Vue/Angular/Svelte)
- ✓ 🗺️ Router (React Router, Vue Router, Angular Router)
- ✓ 🏪 State Manager (Redux, Zustand, Pinia, NgRx)
- ✓ 🪝 Hooks/Composables/Services
- ✓ 💉 Dependency Injection (per Angular)
- ✓ 🎨 Componenti UI (Tailwind, Chakra, DaisyUI, ...)
- ✓ ❓ Altre librerie accessorie

Google Developer Groups

Per la build, poi...

- 🔨 Build scripts
- 🌳 Tree shaking
- 📦 Code splitting
- 🗜️ Minification
- 🗺️ Source maps

Google Developer Groups

Infine, per il deploy...

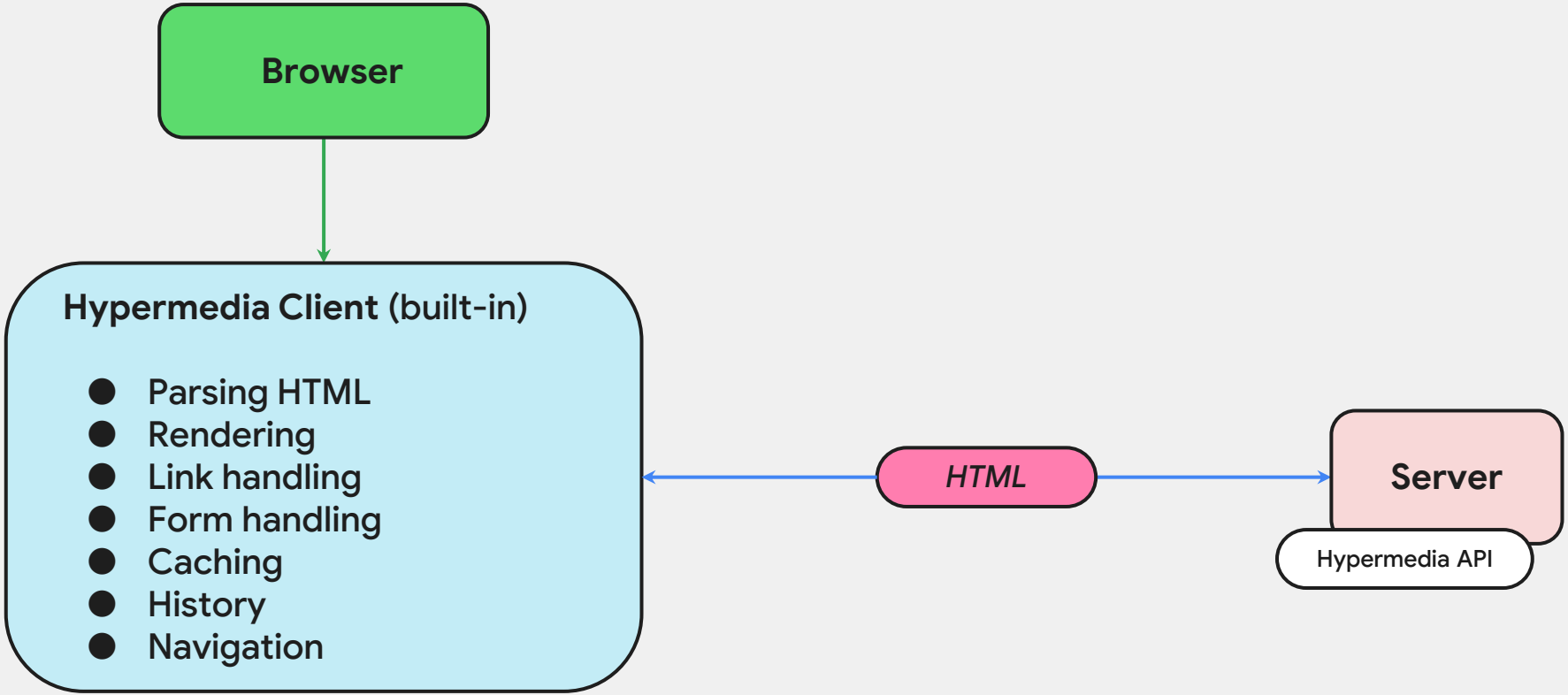
- 🐳 Docker (opzionale, raccomandato)
- ☁️ CI/CD pipeline
- 🌐 CDN configuration

Oh, manca ancora il **backend**!

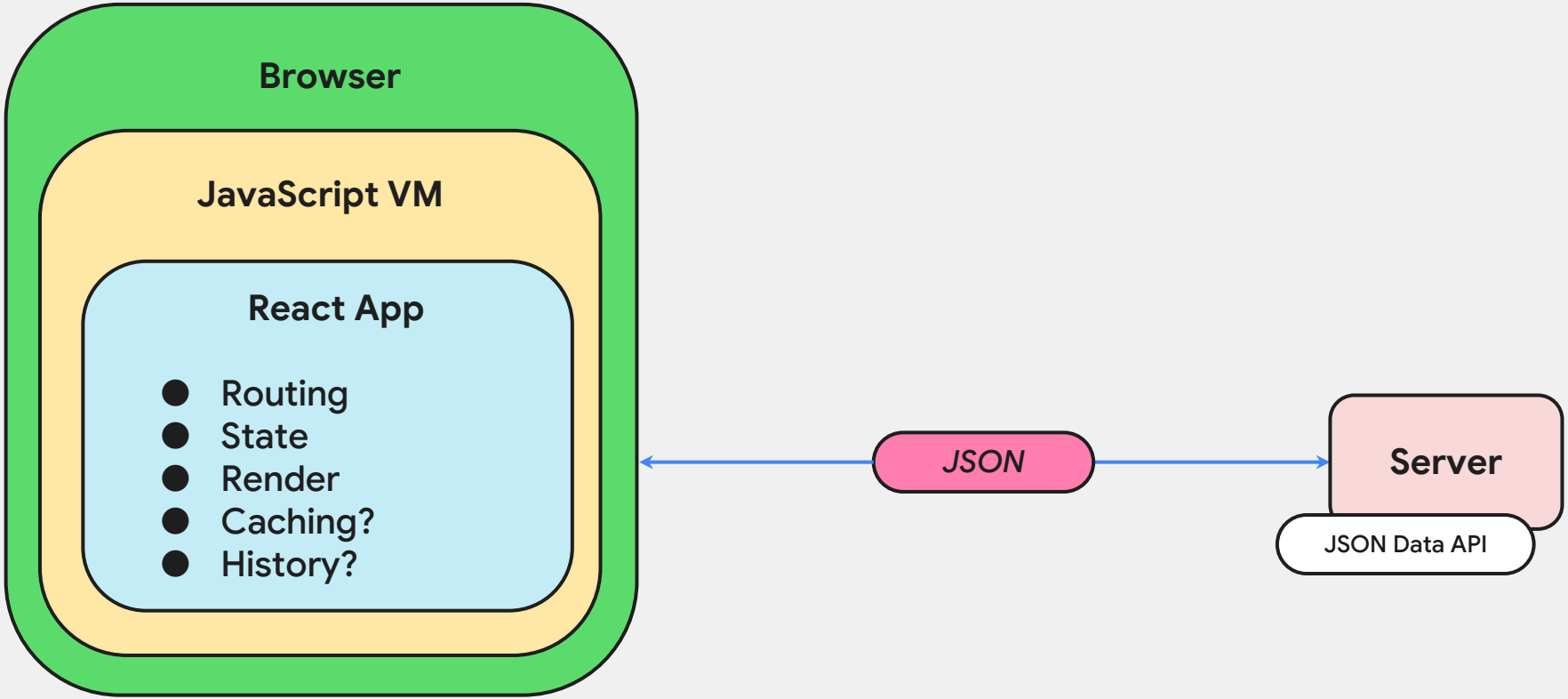
Ma quello è un altro talk... 😄

Google Developer Groups

Prima - Hypermedia (Web 1.0)



Oggi - Single Page Applications (SPA)



Ma perché tutto questo?

- HTML si è fermato, mentre le aspettative degli utenti (e degli sviluppatori) crescevano
- Non è stata una scelta architetturale, ma una necessità data da una limitazione.

E se quella limitazione potesse essere rimossa? 🤔

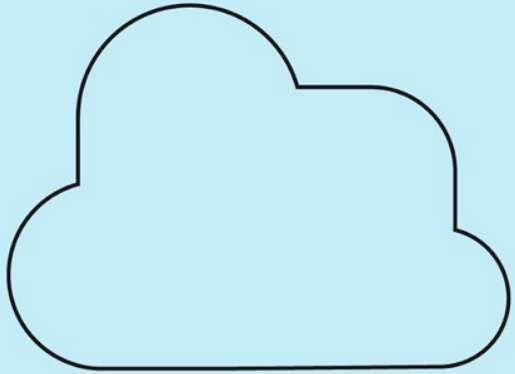
Google Developer Groups

DevFest

Modena

REST API: una promessa mancata?

E se la tua API REST non fosse davvero “**RESTful**” come credi?



Google Developer Groups

Roy Fielding

- Uno dei principali architetti del **protocollo HTTP**
- Co-fondatore del progetto **Apache HTTP Server**
- Ideatore dell'architettura **REST**
- Ha coniato il termine **HATEOAS** (Hypermedia As The Engine of Application State)

Google Developer Groups

"REST è una **architettura di rete**, un modo di intendere i sistemi distribuiti."

— Roy Fielding, 2000

Google Developer Groups

I “constraint” dell’architettura REST

- **Architettura client/server**, come quella tra browser e web server
- **Stateless**: ogni richiesta contiene tutte le informazioni necessarie per poter dare una risposta
- **Caching**: ogni risposta per richieste della stessa risorsa è potenzialmente archiviabile in memoria
- **Uniform interface**: sono i componenti del “cuore” del sistema REST
  - Identificazione delle risorse (URL), rappresentazione (header, negoziazione), self-descriptive (tutto è nel messaggio)
- **Layered System**: consente a più server di fare da intermediari (apertura a proxy, CDN, caching distribuito)
- **Scriptable**: codice (opzionale) da eseguire per estendere il client (download di feature)

Google Developer Groups

Self-Descriptive Message

```
<html lang="en">
<body>
  <h1>Joe Smith</h1>
  <div>
    <div>Email: joe@example.com</div>
    <div>Status: Active</div>
  </div>
  <p>
    <a href="/contacts/42/archive">Archive</a>
    <a href="/contacts/42/edit">Edit</a>
  </p>
</body>
</html>
```

- ✅ Dati presenti
- ✅ Controlli presenti (link)
- ✅ Azioni possibili indicate chiaramente
- ✅ Il browser sa esattamente cosa fare

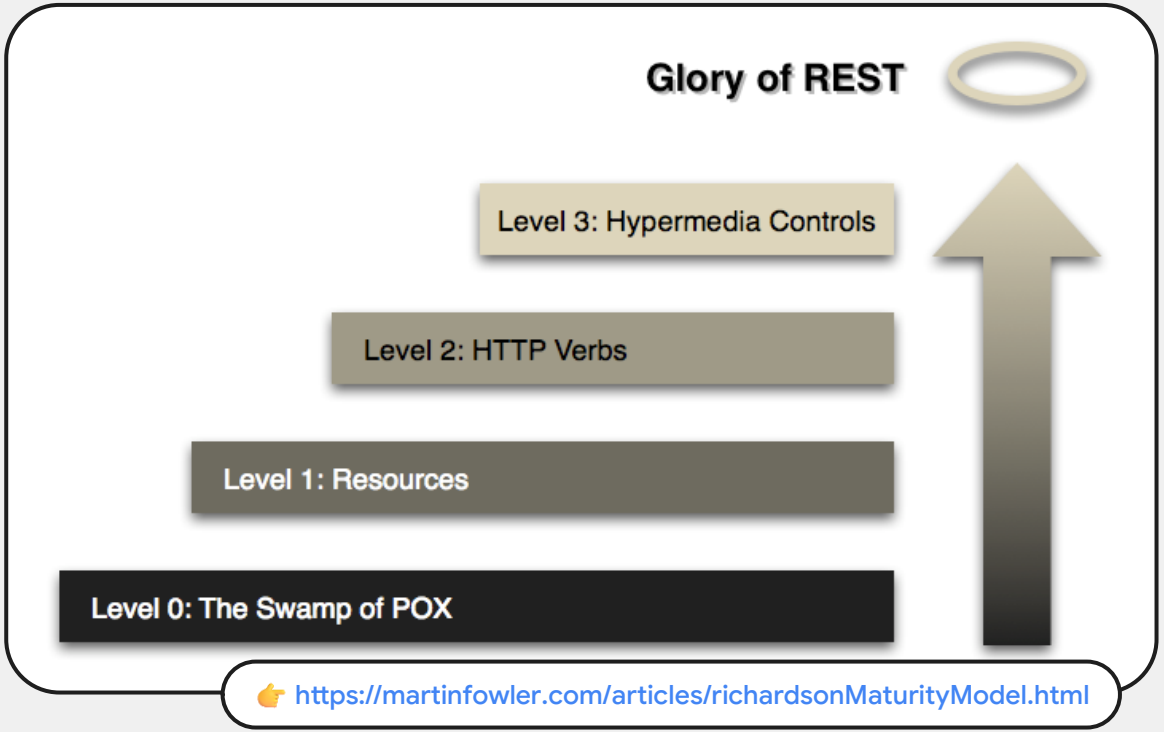
NON Self-Descriptive Message

```
{
  "name": "Joe Smith",
  "email": "joe@example.com",
  "status": "Active"
}
```

✅ Dati presenti
❌ Nessun controllo
❌ Azioni possibili: ??? (dove sono?)
❌ Il client deve già sapere cosa fare

Richardson Maturity Model

Steps toward the glory of REST



"I am getting frustrated by the number of people calling any HTTP-based interface a REST API. That is RPC. It screams RPC. [...] If the engine of application state is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period." 😤

— *Roy Fielding, 2008* (Fonte)

Google Developer Groups

HATEOAS: il principio dimenticato

```
<h1>Joe Smith</h1>
<div>Status: Active</div>
<p>
  <a href="/contacts/42/archive">Archive</a>
  <a href="/contacts/42/message">Message</a>
</p>
```

Step 1 - **Contatto attivo**

HATEOAS: il principio dimenticato

```
<h1>Joe Smith</h1>
<div>Status: Archived</div>
<p>
  <a href="/contacts/42/unarchive">Unarchive</a>
</p>
```

Step 2 - **Contatto archiviato**

HATEOAS

- Il link “Archive” scompare
- Appare “Unarchive”
- Il browser non sa cosa sia un contratto archiviato, né come vada gestito...
- ...tuttavia sa come presentare i controlli necessari

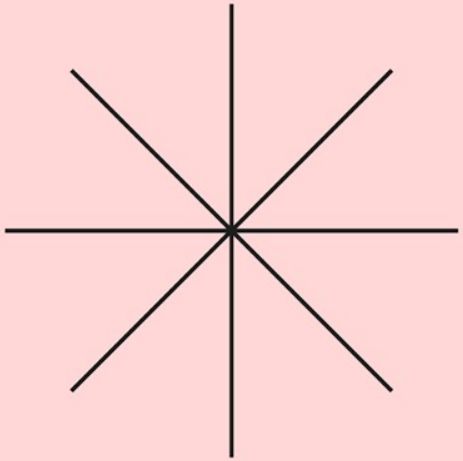
VS

JSON

- Il client riceve `{"status": "Archived"}`
- Deve sapere cosa significa l’informazione
- Deve sapere quali azioni sono disponibili
- La logica è duplicata tra client e server

{ DevFest }

Editable Location



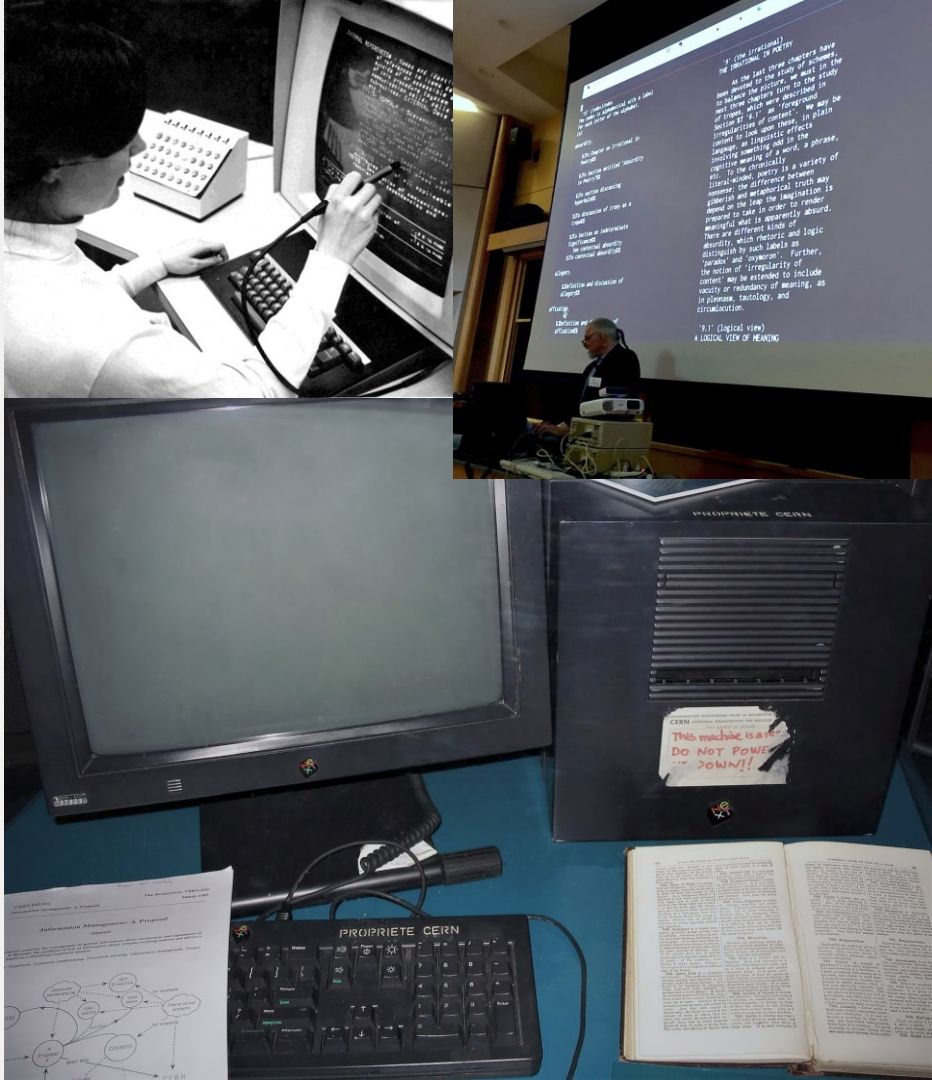
Hypermedia System: un concept vecchio, anzi nuovo!

Il **browser** e il **World Wide Web** sono di nuovo al centro!

Google Developer Groups

Che è ‘sto “hypermedia”?

- Nella definizione più canonica, è il World Wide Web
- E’ un sistema che aderisce all’architettura RESTful di Roy Fieldings
- Comprende un set di componenti specifici



Google Developer Groups

Hypermedia

E’ un ipertesto con navigazione non lineare (documenti collegati tramite link), ovvero **HTML**!

- ✓ Generalmente fruibili in modo ottimale solo da un computer (o equivalente)
- ✓ Può - anzi deve - contenere altri tipi di media, come immagini, video, ecc.

Google Developer Groups

Hypermedia Control

E’ un elemento dell’hypermedia che descrive (o regola) un certo tipo di iterazione (es. un collegamento)

- ✓ E’ ciò che distingue questo tipo di media da tutti gli altri

Google Developer Groups

Hypermedia Server

Server che espongono API capaci di generare hypermedia, quindi testi, video, immagini, ecc.

- ✓ Si avvalgono di protocolli di comunicazione idonei allo scopo, come HTTP

Google Developer Groups

Hypermedia Client

Programma che supporta i protocolli e consente di fruire di contenuto espresso come hypermedia

✓ In poche parole, qualunque browser moderno

Google Developer Groups

HDA (Hypermedia-Driven Applications)

Hypermedia-Driven Application (HDA) è un'applicazione web che usa (e scambia) hypermedia come meccanismo primario di comunicazione con il server.

- Approccio estremamente semplice nella creazione delle applicazioni web
- Utilizzo formalmente corretto degli elementi semantici di HTML (no alla soup!)
- Estrema tollerabilità ai cambiamenti dell'API lato server
- Possibilità di sfruttare senza sforzo le caratteristiche native dei browser (es. caching)
- Effort minore per il team nel realizzare e gestire lo sviluppo del sistema software
- Mitigazione della cosiddetta "JavaScript Fatigue" e della "SPA Fatigue"
- Permeabilità totale alla SEO (Search Engine Optimization)
- Compatibilità all'indietro con i browser estremamente elevata

Quando scegliere un approccio HDA?

Hypermedia è spesso - non sempre - un'ottima scelta per una web application!

- Applicazioni che non richiedono necessariamente particolare interattività dell'utente
  (vedi esempi come Amazon, eBay, siti di notizie, shopping, forum, ecc.)
- Quando la maggior parte del valore è aggiunto server-side
  (ad esempio, report o analisi di dati che devono essere presentati all'utente)
- Quando devi fare delle banali operazioni CRUD su una base dati
  (ad esempio, creare un frontend per un database, un login, un form di sondaggi, ecc.)
- Non è richiesta logica di routing complessa client-side o gestire modelli dati sul client
  (in breve, quando il tasto «Indietro» o il back-linking semplicemente funzionano)

Quando NON scegliere l'approccio HDA?

...ovvero, quando è meglio costruire la classica Single Page Application (SPA).

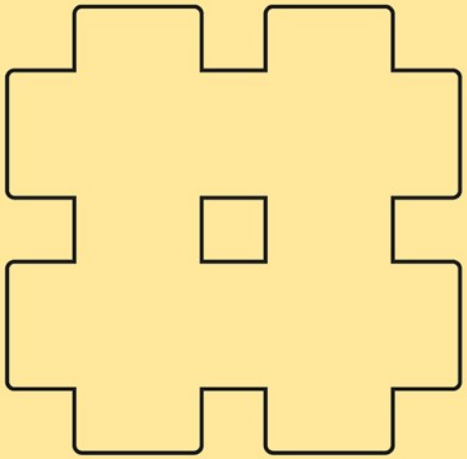
- L'interazione con l'utente è frequente e permea ogni frangente dell'applicazione (ad esempio, uno spreadsheet richiede feedback a ogni pressione di tasto)
- Vi è una integrazione stretta necessaria tra applicazione e API del browser (es. nei casi in cui la libreria/framework JS offre già qualche integrazione)
- Il numero di utenti è estremamente elevato o vi è suscettibilità al consumo di risorse
- Le feature richieste sono complesse: non si possono rifiutare per cose più semplici (ad esempio, quando il cliente forza la mano sull'attuazione di alcune funzionalità)
- Il team è «suscettibile» all'utilizzo di specifici framework e librerie, o ha esperienze specifiche sfruttabili per ridurre il Time-to-Market

Tutto bello, ma...
come superare le limitazioni di HTML?

Google Developer Groups

{ DevFest }

Editable Location



La libreria <htmx>: l'HTML così come avrebbe dovuto essere!

E così possiamo finalmente creare le **Hypermedia-Driven Applications** (*HDA*)!

Google Developer Groups

L’implementazione che conosciamo...

- Azione delegata a codice JavaScript
- La comunicazione avviene tramite AJAX o Fetch API
- E’ il runtime di JavaScript a gestire il processo
- L’API lato server è una semplice JSON Data API
- Il metodo che aggiorna la UI deve conoscere la struttura dati

```
<button onclick="fetch('/api/v1/contacts/1') \
                 .then(response => response.json()) \
                 .then(data => updateUI(data))">
  Click to fetch
</button>
```

Primo esempio di HDA (con <htmx/>!)

- Non abbiamo codice JavaScript: usiamo gli attributi
- La risposta del server è attesa in formato HTML, e non JSON
- L'interazione è simile a quella con elementi <a> e <form>
- Non occorre codice di aggiornamento della UI
- Il browser (e la libreria <htmx>) si occupano delle interazioni

```
<button hx-get="/contacts/1" hx-target="#contact-ui">
  Click to fetch
</button>
```

Google Developer Groups

<HTMX/> alla riscossa!

</>

htmx

- Estendere HTML, non sostituirlo
- Libreria più “Hypermedia-oriented”
- Matura, stabile, piccola (~14kb), versatile, zero dipendenze
- Non richiede JavaScript (al developer)
- Puoi usarlo con qualsiasi linguaggio e tecnologia lato server

👉 https://htmx.org/

Google Developer Groups

Richieste HTTP con HTMX

- Attributi per eseguire richieste HTTP
  - `hx-get`, `hx-post`, `hx-put`, `hx-patch`, `hx-delete`
- Ognuno di questi attributi indica ad HTMX il comando da usare per la richiesta HTTP al verificarsi di un evento
- La libreria aggancia automaticamente il codice JavaScript per effettuare la chiamata

```
<button hx-get="/tasks">
  Load tasks
</button>
```

E’ solo HTML!

- HTMX si attende risposte in formato HTML
- Ogni hypermedia control ottiene HTML come risposta alla chiamata
- Le risposte possono essere “parziali” o complete
  - Non andremo a rimpiazzare l’intero documento
  - Aggiungeremo o sostituiremo contenuti all’interno del documento esistente (dove? con `hx-target`)
  - Ottimizzazione banda, risorse, tempi di elaborazione: efficienza!

Più integrazione!

- Possiamo decidere la modalità di scambio
  - `hx-swap` (*innerHTML, outerHTML, beforebegin, beforeend, afterbegin, afterend*, ...)
- Possiamo scegliere l'evento a cui rispondere
  - `hx-trigger` (*click, change, mouseenter*, ...)
- Possiamo inviare valori di input sfruttando le "enclosing form" (oppure `hx-include`, `hx-vals`, ...)
- Supporto alla History del browser (con l'attributo `hx-push-url`)

```
<form action="/contacts" method="get" class="toolbar">
  <label for="search">Search Term</label>
  <input id="search" type="search" name="q"/>
  <button hx-post="/contacts" hx-target="#main">
    Search
  </button>
</form>
```

Bonus: l’accessibilità!

- **Single Page Applications**
  - ❌ La navigazione da tastiera è spesso “rotta”
  - ❌ Gli screen reader sono spesso confusi dai cambiamenti dinamici
  - ❌ Serve spesso codice extra, librerie dedicate, una expertise specifica
- **Hypermedia (e HTML semantico)**:
  - ✅ La navigazione da tastiera è attiva
  - ✅ Gli screen reader funzionano naturalmente
  - ✅ Progressive Enhancement naturale (remember “Unobtrusive JavaScript”?)
  - ✅ Back/Forward perfettamente funzionante

Non stai “aggiungendo” accessibilità: stai semplicemente non rompendo quella che il browser ti da. Gratis.

Google Developer Groups

Takeaways

"Hypermedia-Driven Applications non sono nostalgia. Sono web-native, semplici, RESTful e accessibili. Per molte delle applicazioni web moderne sono la scelta migliore."

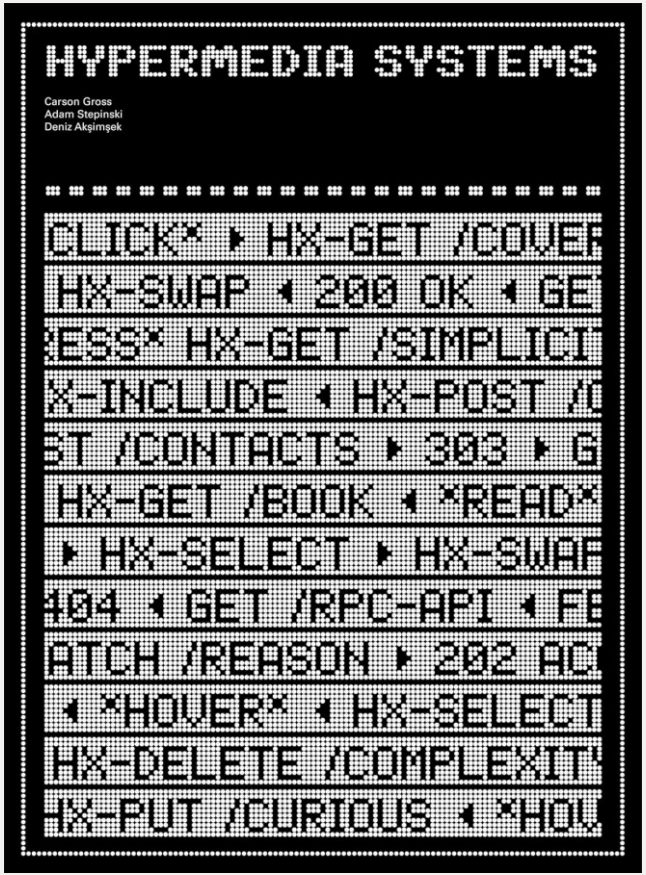
Google Developer Groups

Per saperne di più...

Hypermedia Systems

di Carson Gross, Adam Stepinski, Deniz Akşimşek

Learn how hypermedia, the revolutionary idea that created The Web, can be used today to build modern, sophisticated web applications today, often at a fraction of the complexity of popular JavaScript frameworks.



Google Developer Groups

Q & A

DevFest

Modena

Thanks! 

feedback