



DelphiDay  
italian conference

# Hypermedia Driven Apps

*seminario*

Scopriamo cosa sono e come crearle con Delphi e HTMX



# MARCO BREVEGLIERI

 Software Developer |  Trainer and Consultant |  Tech Content Creator

 [www.breveglieri.it](http://www.breveglieri.it)

 [marco.breveglieri@abls.it](mailto:marco.breveglieri@abls.it)

 [twitter.com/mbreveglieri](https://twitter.com/mbreveglieri)

 [github.com/marcobreveglieri](https://github.com/marcobreveglieri)

 [linkedin.com/in/marcobreveglieri](https://linkedin.com/in/marcobreveglieri)



19-20 Giugno 2025  
Piacenza





# OPEN-SOURCE PROJECTS

[github.com/marcobreveglieri](https://github.com/marcobreveglieri)



**CompilaQuindiVa**

[www.twitch.tv/compilaquindiva](https://www.twitch.tv/compilaquindiva)

+  **YouTube**



**Delphi Podcast**

[www.delhipodcast.com](https://www.delhipodcast.com)



**Prometheus Client for Delphi**

[github.com/marcobreveglieri/prometheus-client-delphi](https://github.com/marcobreveglieri/prometheus-client-delphi)



19-20 Giugno 2025  
Piacenza





# AGENDA

---

- ✓ **Prima di iniziare...**
- ✓ **Hypermedia: le origini**
- ✓ **HDA: Hypermedia-Driven Apps**
- ✓ **Dalla teoria alla pratica**
- ✓ **HDA con la libreria HTMX**
- ✓ **Recap + Q & A**



Prima di  
iniziare...

0



# Lo sviluppo web oggi

---

- Molti sviluppatori «gggiovani» che fanno frontend conoscono solamente SPA (Single Page Applications)
- Non hanno utilizzato altre librerie diverse da quelle più blasonate (Angular, React.JS, Vue, Svelte, ...)
- Vedono il server solo come un fornitore di dati JSON e nient'altro
- In particolare, il linguaggio HTML in sé (non solo per gli elementi, ma per il suo essere «hypermediale») è una «cosa da boomer»
- Hypermedia? Ha ancora un senso! E tanto! 🙌



# Quali sono i «problemi»?

---

- Molti sviluppatori ignorano o trascurano le «funzionalità Hypermedia»
- Costruiscono applicazioni quasi interamente utilizzando solo JavaScript
  - Non si naviga tra pagine (o almeno, non nel modo classico)
  - L'interfaccia è aggiornata solo con codice JavaScript
  - Si comunica con un backend quasi esclusivamente con JSON e via AJAX
  - Sono tipicamente delle SPA con una libreria o framework per frontend
- HTML diventa quasi un linguaggio che «si usa perché sta lì da sempre»
- Queste applicazioni non sono «hypermedia driven»
  - Non sfruttano l'«hypermedia system» del web!

# Una saggia citazione

---



«IL FUTURO NON È STATO SCRITTO.  
NON ESISTE DESTINO SE NON  
QUELLO CHE CREIAMO PER NOI  
STESSI»

*KYLE REESE – TERMINATOR 2*





# Ma che è 'sta Hypermedia? 🤔

---

- Parliamo del linguaggio HTML, ma non solo di questo
- Del protocollo HTTP, della sua abilità di trasferire HTML
  - Header di richiesta e di risposta
  - Gestione della cache
  - Response Code
  - ecc.
- Dell'«Hypermedia Client» per eccellenza: il browser!
  - Ambiente di esecuzione completo, con un potente runtime JavaScript (e anche WASM!)
  - Oggi esegue applicazioni «thick client», quasi come quelle desktop



# Che cos'è un Hypermedia System?

---

- Nella definizione più canonica, è il World Wide Web
- Precisamente, lo definiamo come un sistema che aderisce alla «RESTful network architecture» teorizzata da Roy Fieldings
- REST spesso vuol dire JSON, ma...
  - JSON non è un «hypermedia» naturale: è privo di «Hypermedia Controls»
  - Se stai pensando a JSON, dimentica ciò che conosci: vedi tutto con occhi nuovi
- Ricorda: ciò che Roy Fieldings considera REST nasce nel WWW di fine anni '90!
  - A quel tempo, i browser web scambiano solo hypermedia (ossia testo, link, controlli, form, ecc.)



# Riscopriamo il linguaggio HTML

---

- Ha un ruolo centrale nella storia che stiamo per raccontare
  - Miliardi di utenti usano sistemi basati su HTML e HTTP ogni giorno
- E' «hypermedia friendly» quando ne esprime le caratteristiche:
  - Veicola lo stato dell'applicazione agli utenti che la visualizzano nel browser
  - Veicola informazioni salienti agli screen reader che leggono i siti (importante!)
  - Espone ciò che i motori di ricerca estraggono per l'indicizzazione del sito
  - Descrive in modo chiaro il comportamento dell'applicazione agli sviluppatori
- Quando è «buono», fa fare al browser gran parte del lavoro!







# Hypermedia: le origini

1



# Hyper-glossario di hyper-termini

---

- **Hypermedia**: tipo di media (es. ipertesto) con navigazione non lineare (ad esempio, documenti collegati tramite link)
  - Generalmente fruibili in modo ottimale solo da un computer (o equivalente)
  - Può – anzi deve – contenere altri tipi di media, come immagini, video, ecc.
- **Hypermedia Control**: un elemento dell'hypermedia che descrive (o regola) un certo tipo di interazione (es. un collegamento)
  - E' ciò che distingue questo tipo di media dagli altri
- **Hypermedia Server**: server che espongono API capaci di generare hypermedia, quindi testi, video, immagini, grazie ad appositi protocolli (es. HTTP)
- **Hypermedia Client**: programma che supporta i protocolli e consente di fruire del contenuto hypermedia (ad esempio, uno dei tanti web browser moderni)



# I primi hyper-passi...

- **1945:** Vannevar Bush (direttore dell'Office of Scientific Research and Development) pubblica il saggio «As We May Think», dove si teorizza il funzionamento del Memex
- **1963:** Ted Nelson, pioniere dell'IT, lavora a un Hypertext Editing System (HES) alla Brown University e crea FRESS (File Retrieval and Editing System)
  - Si tratta della prima apparizione digitale della funzione di «undo»
  - Usato per documentare la missione Apollo



[https://commons.wikimedia.org/wiki/File:HES\\_IBM\\_2250\\_Console\\_grlloyd\\_Oct1969.png#/media/File:HES\\_IBM\\_2250\\_Console\\_grlloyd\\_Oct1969.png](https://commons.wikimedia.org/wiki/File:HES_IBM_2250_Console_grlloyd_Oct1969.png#/media/File:HES_IBM_2250_Console_grlloyd_Oct1969.png)



# Una hyper...evoluzione!

→ 1968: Douglas Engelbart (Stanford Research Institute) mostra un sistema ipermediale completo realizzato a Menlo Park (a.k.a. «The Mother of All Demos»)

- Chat video e audio
- Una UI basata su finestre
- Testo cliccabile
- Prototipo di mouse
- Help context-sensitive
- Controllo di versione
- Accenno di WYSIWYG
- Linguaggio compilabile

P.S.

alcuni membri del team finirono allo Xerox Palo Alto Research Center...  
vi ricorda niente? 😊

By Gregory Lloyd - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=79746873>



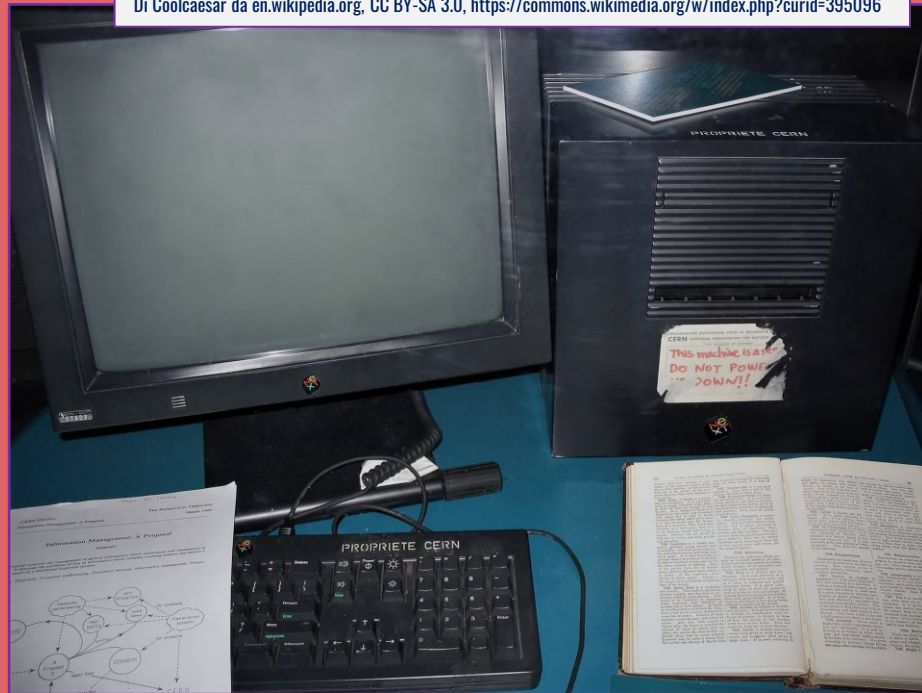




# La nascita del Web

- 1990: Tim Berners Lee pubblica il primo sito web al CERN
  - Ha lavorato all'idea dell'ipertesto per 10 anni
  - Le tecnologie per realizzarlo erano già tutte lì
  - Senza di esso, la condivisione tra ricercatori era diventata insostenibile
  - Individua il momento giusto per ottenere il supporto istituzionale necessario a dare vita al web
  - Il lavoro principale svolto consiste nell'aggregare tutto l'esistente (sistemi, protocolli, ecc.) e standardizzarli
- 1994: Tim fonda il W3C!

Di Coolcaesar da en.wikipedia.org, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=395096>





# Dal Web alle API REST

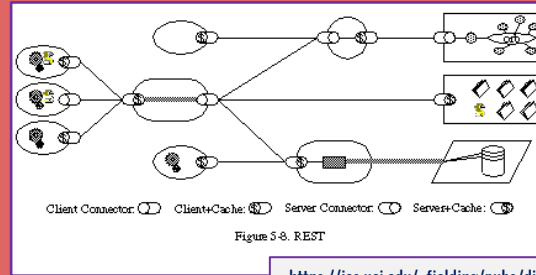
→ 2000: Roy Fielding pubblica la tesi «Architectural Styles and the Design of Network-based Software Architectures»

→ Roy è uno dei principali architetti del protocollo HTTP

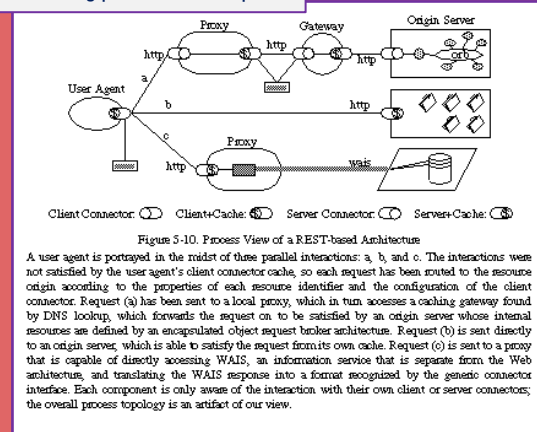
→ E' co-fondatore del progetto Apache HTTP Server e membro di Apache Foundation

→ Ha coniato HATEOAS (Hypermedia As The Engine Of Application State), principio cardine dell'architettura REST (spesso dimenticato)

→ HATEOAS si contrappone a IDL, CORBA, SOAP e a tutti gli altri protocolli simili!



<https://ics.uci.edu/~fielding/pubs/dissertation/top.htm>





# HTML evolve con il Web

- 1.0: versione «read only» per pubblicare documenti accademici tramite collegamenti ipertestuali
- 2.0: grazie al tag `<form>`, viene fornito un meccanismo per aggiornare i dati oltreché leggerli
  - Transizione da linguaggio document-oriented ad application-oriented
- 3.0: si aggiungono tabelle, gestione del testo attorno alle figure, mai adottato ma precursore del 4.0
- 4.0+ (anno 1999): fogli di stile, oggetti embedded, accessibilità
- 5.0+ (anno 2008): elementi semantici, nuovi media integrati, grafica vettoriale e bitmap, validazione, tantissimo altro!





# Cosa NON è hypermedia, di nuovo

→ L'azione è delegata a codice JavaScript e non al linguaggio ipermediale supportato dal client

- Viene inviata la richiesta al server tramite *fetch()*
- La risposta viene recuperata con un oggetto
- Il contenuto viene convertito dal formato JSON
- Viene invocata una funzione per aggiornare l'interfaccia utente

→ La comunicazione avviene tramite AJAX dietro le quinte

- Benché la «X» stia per «XML», si usa comunque principalmente JSON

→ E' il runtime di JavaScript a dover gestire il processo

→ Aspetto cruciale: la comunicazione via JSON non usa hypermedia (non ci sono né «hyper controls» né media)

- Si tratta solo di una Data API, poco di più
- Il metodo `updateUI()` deve conoscere la struttura dati: se essa cambia, anche il codice deve farlo (coupling!!)

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







# Single Page Application (SPA)

---

- Aggiornare a mano la pagina è faticoso: la china termina con la creazione di una SPA
  - Non si naviga più con Hypermedia Controls, ma programmaticamente con JavaScript
  - Si utilizza una libreria per astrarre la parte di aggiornamento della UI in base al modello dati
    - La maggior parte adotta un meccanismo di binding «a due vie»
  - L'uso delle librerie più conosciute richiede competenze specifiche quando le esigenze crescono
    - Ci si lega a framework tendenzialmente complessi per progetti medio/grandi
    - Si utilizzano molte dipendenze che devono poi essere regolate
    - Si utilizzano tool di sviluppo aggiuntivi (es. transpiler, package manager, ecc.)
  - La logica viene spesso duplicata (esiste sul server e in parte sul client)
- Le SPA diventano sostanzialmente dei «thick client» con ostacoli aggiuntivi
  - Linguaggio diverso dal server (a volte), build grandi, ampio TTFB, hydrating, SEO, SSR



# Perché Hypermedia è rimasto indietro?

- Come mai lo sviluppo non è proseguito con l'uso dell'Hypermedia?
  - Le esigenze degli utenti web e la richiesta di interattività «come sul desktop» è cresciuta esponenzialmente
  - Il linguaggio HTML e la sua espressività non ha seguito di pari passo questa forte richiesta
  - La UX fruibile con JavaScript ha decretato la «vittoria» delle SPA come architettura per le applicazioni web
- Parere di alcuni movimenti (es. Lean Web) è che «non avrebbe dovuto andare così»
  - L'industria del web avrebbe dovuto richiedere più interattività per il linguaggio HTML
  - L'approccio SPA era più familiare rispetto ad altre metodologie di sviluppo
- E' prevista una «resurrezione» dell'Hypermedia?
  - Cosa sarebbe successo con una evoluzione del linguaggio HTML?
  - Quali nuovi elementi e controlli ipermediali sarebbero stati introdotti?
  - Sarebbe stato possibile sviluppare applicazioni web in modo diverso oggi? E domani?
- Librerie JavaScript «Hypermedia-Oriented»
  - Esistono nuove librerie che usano JavaScript non per sostituire l'HTML, ma per aumentarne le potenzialità
  - Le librerie JavaScript moderne (come <htmlx>) estendono l'HTML in sé come «linguaggio hypermedia»
  - L'obiettivo è quello di riportare hypermedia al centro dello sviluppo delle applicazioni web



# Hypermedia Driven Apps

# 2



# HDA (Hypermedia-Driven Application)

**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 di applicazioni Web
- Utilizzo formalmente corretto degli elementi semantici di HTML (no alla *<div> soup!*)
- Estrema tollerabilità dei cambiamenti all'API lato server
- Possibilità di sfruttare senza sforzo caratteristiche native dei browser (es. caching)
- Effort minore per il team per realizzare e gestire lo sviluppo del sistema
- Mitigazione della «JavaScript Fatigue» e della «SPA Fatigue»
- Permeabilità totale alla SEO (Search Engine Optimization)
- Compatibilità all'indietro estremamente elevata (ci ricorda qualcosa?) 🤔





# Quando **\*NON\*** scegliere HDA

---

- 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 (in modo particolare nei casi in cui la libreria/framework JavaScript offre già qualche integrazione)
- Il numero di utenti è estremamente elevato, oppure vi è una suscettibilità al consumo di risorse
- Le feature richieste sono articolate e non possono essere «rifiutate» per qualcosa di più semplice (ad esempio, quando il cliente forza la mano sulle modalità di 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



# Quando scegliere 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 un modello dati sul client (in breve, quando il tasto «Indietro» o il back-linking semplicemente funzionano)



# Componenti di un Hypermedia System

- Un **hypermedia**, come il linguaggio HTML
  - Hypermedia Controls: HTML ha nativamente <a> e <form>
  - Uniform Resource Locators (URLs)
- Un **protocollo** di rete, come HTTP(S)
  - Il protocollo HTTP(S) supporta nativamente diversi comandi (GET, POST, PUT, DELETE, ...)
  - Consente di utilizzare gli URL per specificare gli indirizzi delle risorse
  - Determina i server da contattare tramite le parti significative estraibili dagli URL
- Un **server** che esponga una Hypermedia API, ossia che risponda alle richieste usando hypermedia
  - Si presuppone che il server conosca HTTP e risponda utilizzando il linguaggio HTML
  - Si usano codici per informazioni, errori o dirottamenti verso altri URL
- Un **client** che sappia interpretare queste risposte supportando il protocollo di rete



# Primo esempio di HDA (con `<htmx/>`)

```
<button hx-get="/contacts/1" hx-target="#contact-ui">
  Fetch Contact
</button>
```

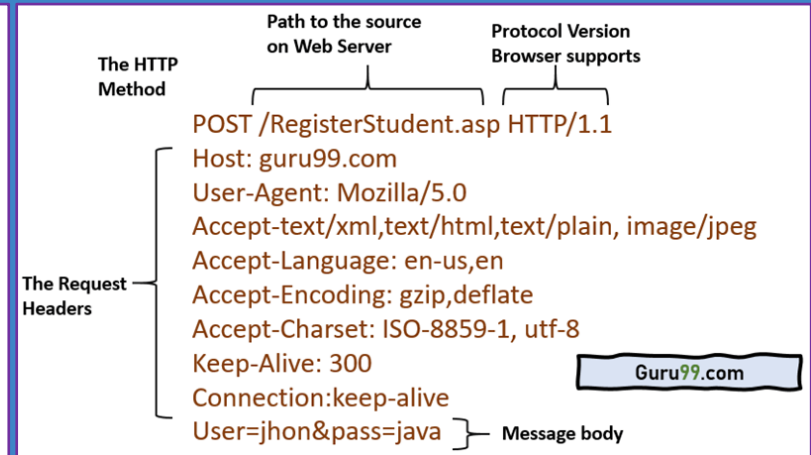
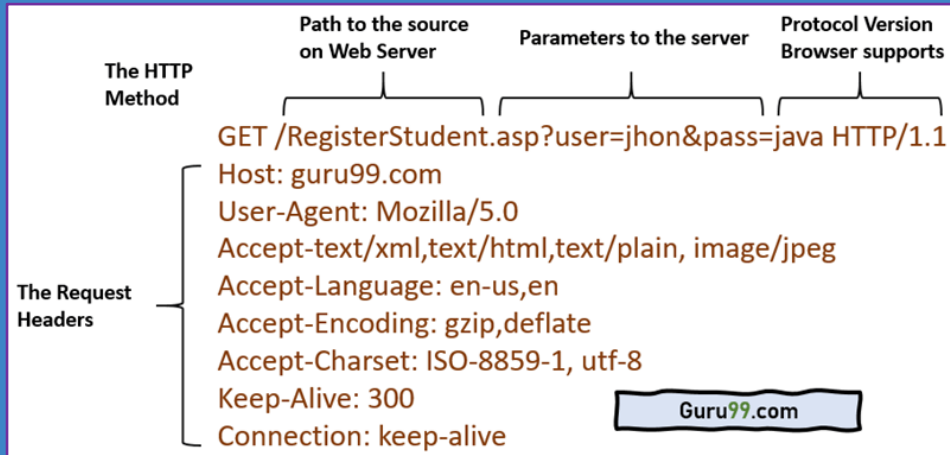
htmx

- NON abbiamo codice JavaScript, ma si usano attributi (approccio dichiarativo)
  - L'attributo `hx-get` dice: quando l'utente clicca sul pulsante, manda una GET all'indirizzo `/contacts/1`
  - L'attributo `hx-target` dice: quando il server risponde, prendi l'HTML e sostituiscilo all'elemento con id `contact-ui`
- La risposta del server è attesa in formato HTML, e non JSON
- L'interazione è del tutto simile a quella con elementi `<a>` o `<form>`



# Protocollo HTTP – Le basi

- Basato su testo, facilmente leggibile e ispezionabile da una persona
- Uso di intestazioni (per richiesta e risposta)





# Protocollo HTTP – Verbs (comandi)

- Supporto di metodi specifici (simili al pattern CRUD)
  - GET: recupera una risorsa, non altera dati sul server, è idempotente
  - POST: invia dati a una risorsa, muta lo stato sul server
  - PUT: sostituisce i dati della risorsa, muta lo stato sul server, dovrebbe essere idempotente
  - PATCH: sostituisce dati parziali della risorsa, muta lo stato sul server, idem come sopra
  - DELETE: elimina la risorsa specificata, muta lo stato sul server
- Il metodo HTTP utilizzato deve corrispondere all'intenzione espressa dall'Hypermedia Control (es. se si usa un link/pulsante per cancellare un elemento, deve inviare un comando HTTP DELETE)
- HTML supporta nativamente solo GET e POST... ma perché mai?? 🙄





# Protocollo HTTP – Status Codes

---

## → 100-199: Informational Responses

- Forniscono informazioni su come il server sta processando la risposta

## → 200-299: Successful Responses

- La richiesta è andata a buon fine!

## → 300-399: Redirection Responses

- Indicano che la richiesta deve essere inviata a un altro URL.

## → 400-499: Client Error Responses

- Il client ha fatto qualche errore nell'invio della richiesta. Sono sistematici.

## → 500-599: Server Error Responses

- Il server ha riscontrato un errore nel tentativo di rispondere alla richiesta.



# Protocollo HTTP – Caching

- Il server può indicare a un client (e a server intermediari) che **una specifica risposta può essere conservata nella cache per le future richieste**
- Caratteristica del protocollo HTTP in sé e caposaldo del paradigma REST
- L'uso di intestazioni nella risposta HTTP possono regolare durata e parametri della cache

👉 <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Caching>

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
Vary: Accept-Language
```



# Hypermedia Server

---

- Ogni linguaggio di programmazione può essere utilizzato per creare un server HTTP
- Qualunque linguaggio ha una pletora di librerie a disposizione per creare queste applicazioni
- Non sei sottoposto alla «pressione» di adottare JavaScript, né sul client né sul server
- Libertà totale nella scelta delle tecnologie sul backend (piattaforme, runtime, librerie, ecc.)
- Supporto diffuso per il markup HTML: ogni linguaggio di programmazione ha un framework
  - Esempio: in Delphi abbiamo WebStencils, TemplatePro, WebBroker (PageProducer)
- Parlando di <htmx/>: la sua community è multi-linguaggio e multi-framework
  - E' usato da pythonisti, delphisti, dotnettisti, rubysti
  - Ciascuna delle community possono dare supporto alle altre
  - Hypermedia (con <htmx/>) come lingua universale per le applicazioni web



# Hypermedia Client

---

Software in grado di capire come interpretare un particolare hypermedia, e i «controlli» che contiene, nel modo corretto.

- L'esempio canonico è il classico **web browser**, che comprende HTML, può presentarlo all'utente e consentire a quest'ultimo di interagire con esso
- Deve sapere come presentare i controlli hypermedia individuati nella risposta del server nella maniera appropriata affinché l'intero sistema funzioni
  - Si tratta del requisito alla base di HATEOAS, implementato raramente dagli sviluppatori (difficoltà elevata!)
  - Difficilmente realizzabile nelle API JSON-based di tipo REST (ecco perché in certi casi si preferisce GraphQL)



# REST (Representational State Transfer)

Nella sua dissertazione, Roy Fielding l'ha pensata come architettura di rete, ossia un modo di intendere i sistemi distribuiti.

- Oggi è associato principalmente a delle JSON Data API
- Ironico: la maggior parte di queste API REST... non lo è
- Nasce nel periodo degli albori del web, in cui le JSON API e lo stesso AJAX non esistevano
- Si riferisce al trasferimento di HTML tramite protocollo HTTP attuato dai primi browser



# REST Constraints

---

Si tratta dei requisiti posti alla base di un sistema cosiddetto «RESTful».

- Architettura client/server, come quella tra browser e web server (es. diversa da CORBA e simili)
- Stateless: ogni richiesta contiene tutte le informazioni necessarie per poter dare una risposta
- Caching: ogni risposta per richieste della stessa risorsa dovrebbe essere archiviabile in memoria
- Uniform interface: requisiti di comportamento dei componenti del sistema, semplice e flessibile
  - Identificazione delle risorse (URL), rappresentazione (header, negoziazione), auto-descrizione (non occorrono altre informazioni esterne per interpretare le risposte, tutto è nel messaggio).
- Layered: consente a più server di fungere da intermediari (meno fondamentale, sebbene utile)
- Scriptabile: il client deve consentire l'esecuzione di codice opzionale (download di feature)





Dalla teoria  
alla pratica!

3



# WebBroker: che cos'è?

---

WebBroker è un framework incluso in Delphi che consente di creare applicazioni e servizi web.

- Versatilità: supporta vari tipi di servizi web (standalone, moduli Apache, ISAPI, ecc.)
- Integrazione: si trova già in Delphi e l'ambiente di sviluppo consente di creare progetti rapidamente
- Scalabilità: le applicazioni possono essere facilmente scalate per gestire requisiti e carichi più elevati

Puoi utilizzare qualunque altro tipo di librerie esistenti di terze parti, sia a pagamento sia open-source (*RAD Server, WiRL, DMVC, MARS, Horse, ...*).



# WebBroker: i componenti principali

---

## → TWebModule

- Componente centrale di un'applicazione WebBroker
- Agisce come contenitore di altri componenti WebBroker
- Gestisce il flusso generale dell'applicazione (azioni)

## → TWebDispatcher

- Responsabile dell'instradamento delle richieste HTTP in arrivo alle azioni

## → TWebActionItem

- Definisce diversi endpoint e come devono essere gestiti
- Ogni azione è associata a un determinato URL
- Tramite gli eventi appositi, è possibile fornire la risposta alla richiesta



# WebBroker: come funziona?

---

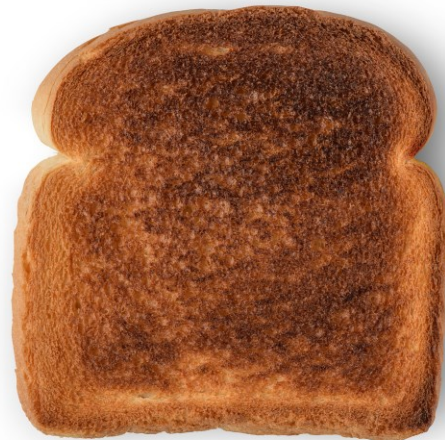
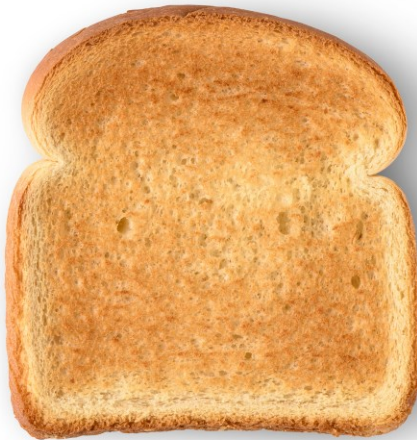
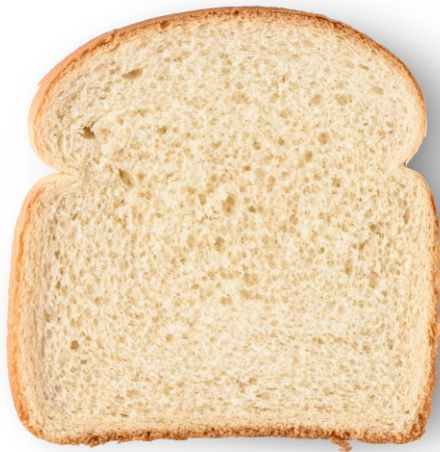
Questo è il **workflow tipico** di funzionamento di un'applicazione web realizzata con WebBroker:

- Arriva una richiesta HTTP alla web application
- Il componente TWebDispatcher instrada la richiesta alla action appropriata
- Il componente TWebActionItem esegue il codice (Delphi) associato
- Viene generata e restituita la risposta al client



# DEMO!

---





# WebStencils: che cos'è?

---

WebStencils è un motore di template che permette l'integrazione e l'elaborazione lato server di file HTML.

- ➔ Introdotta in Delphi 12.2, fornisce un motore di template richiamando l'approccio di ASP.NET Razor
- ➔ La sua natura «agnostica» non impone l'uso di specifiche librerie di frontend per una flessibilità totale
- ➔ Può essere tranquillamente utilizzato anche in altri contesti (es. generazione di testo per e-mail, report, stampe, tutti i contesti in cui può essere impiegato un motore di template)

Se preferite, potete utilizzare anche altri motori di templating sia commerciali sia open source, come ad esempio *TemplatePro*.





# WebStencils: a cosa serve?

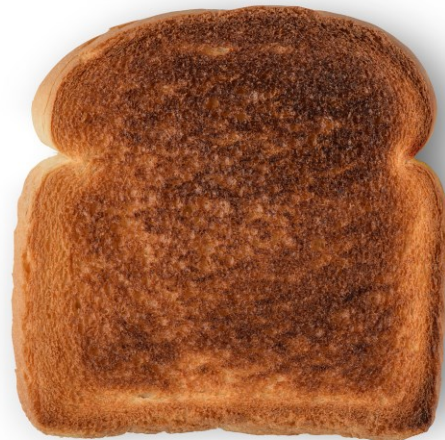
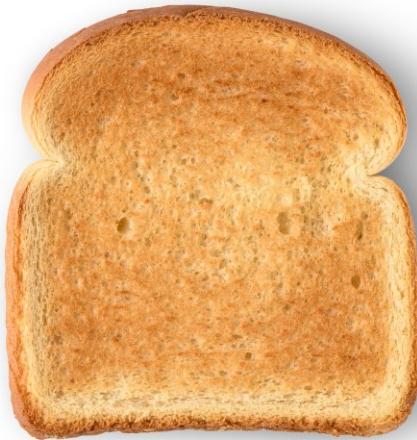
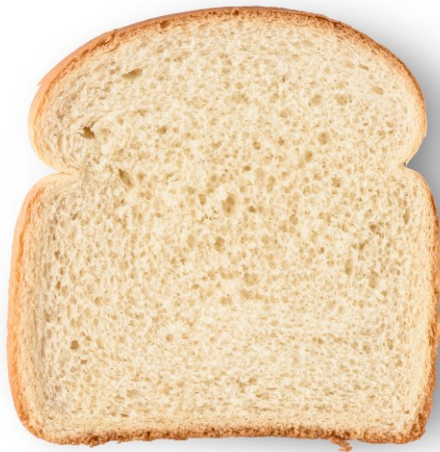
---

- ➔ Sviluppo di siti web con navigazione classica
  - ➔ Permette la creazione di diverse tipologie di siti web, come blog, cataloghi online, sistemi di ordinazione, ecc.
- ➔ Riutilizzo della logica di business esistente
  - ➔ Gli sviluppatori Delphi possono esporre logica di business esistente con template riusabili e personalizzabili
  - ➔ Facilita l'integrazione di progetti pre-esistenti nel contesto del web
  - ➔ Accelera lo sviluppo e la collaborazione tra i componenti del team
- ➔ Miglioramento dell'esperienza di sviluppo
  - ➔ Mira a semplificare il flusso di lavoro e aumentare la flessibilità
  - ➔ Riduce la complessità del codice frontend, leggendolo più leggibile, facile da debuggare e mantenere
  - ➔ Può essere utilizzato in combinazione con librerie JavaScript come HTMX, che è quella che utilizzeremo!



# DEMO!

---





# Cosa abbiamo fatto?

## Classica applicazione Web 1.0 con diversi pregi...

- E' robusta e usa le tecnologie native
- E' semplice da comprendere
- Uso del pattern Post/Redirect/Get
- Domain Logic e CRUD nel «Controller»
- Implementazione REST a tutti gli effetti (usiamo REST e HATEOAS)
  - La nostra applicazione è molto più RESTful del 99% delle JSON API in giro!
- Percepita come «legacy»

ma non possiamo usare tutte le feature  
HTML/HTTP! 🙄

## ...ma ha anche dei difetti... 😞

- Refresh dell'intera pagina a ogni richiesta/interazione
- Flicker quando si naviga tra le pagine
- Si perde lo «scroll state» (punto in cui ci si trovava prima del refresh)
- Necessità di muoversi nella UI alla ricerca dei dati e dei comandi
- Non sfrutta appieno tutti i comandi HTTP (solo GET e POST)



The image features two hands, palms up, holding pills. The left hand holds a green pill, and the right hand holds a red pill. A blue callout bubble points to the green pill, and a red callout bubble points to the red pill. A red bracket connects the two callouts. The background is dark and out of focus.

JavaScript  
(SPA)

Hypermedia-Driven  
(HDA)



# HDA con la libreria HTMX!

# 4



# Riflettiamo assieme...

---

- **Perché solo gli «anchor» e i «form» fanno da controlli?**
  - HTML dovrebbe essere esteso per consentire a qualsiasi elemento di effettuare richieste al server e diventare un hypermedia control
- **Perché solo gli eventi «click» e «submit»?**
  - HTML dovrebbe essere esteso per consentire a qualunque evento di scatenare richieste HTTP
- **Perché solo GET e POST?**
  - HTML dovrebbe consentire di utilizzare tutti i comandi HTTP
- **Perché dobbiamo aggiornare tutta la pagina?**
  - HTML dovrebbe consentire di aggiornare specifici elementi e non necessariamente l'intero documento



# HTMX alla riscossa!

- Libreria che estende HTML
- Non è l'unica libreria JavaScript esistente di questa tipologia
  - È quella più «pura» per l'estensione di HTML come Hypermedia
- L'ultima versione disponibile (al momento) è la 2.0
- Si può usare da CDN o installare come package con NPM
- Non richiede JavaScript (?)
  - Usa attributi per guidare il comportamento







# 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: «Quando fai clic (o altro) su questo elemento, fai una richiesta HTTP del tipo richiesto dall'attributo»
- La libreria aggancia automaticamente il codice JavaScript per effettuare la chiamata AJAX

```
<button hx-get="/tasks">  
  Carica i task  
</button>
```



# E' solo HTML!

- HTMX si attende risposte in formato HTML
- Ogni hypermedia control ottiene HTML come risposta alla chiamata AJAX
  - AJAX è Asynchronous JavaScript & XML, ricordi?
- Le risposte possono essere «HTML parziali»
  - Non andremo a rimpiazzare l'intero documento
  - Includeremo/sostituiremo contenuti all'interno del documento esistente (dove? **hx-target**)
  - Ottimizzazione banda, risorse, tempi di elaborazione: efficienza!

```
<div id="main">  
  <button hx-get="/contacts" hx-target="#main">  
    Get The Contacts  
  </button>  
</div>
```



```
<ul>  
  <li><a href="mailto:joe@example.com">Joe</a></li>  
  <li><a href="mailto:sarah@example.com">Sarah</a></li>  
  <li><a href="mailto:fred@example.com">Fred</a></li>  
</ul>
```



```
<div id="main">  
  <ul>  
    <li><a href="mailto:joe@example.com">Joe</a></li>  
    <li><a href="mailto:sarah@example.com">Sarah</a></li>  
    <li><a href="mailto:fred@example.com">Fred</a></li>  
  </ul>  
</div>
```



# 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`, ecc.)
- Supporto alla history del browser (`hx-push-url`)

👉 Reference: <https://htmx.org/reference/>

👉 Esempi: <https://htmx.org/examples/>

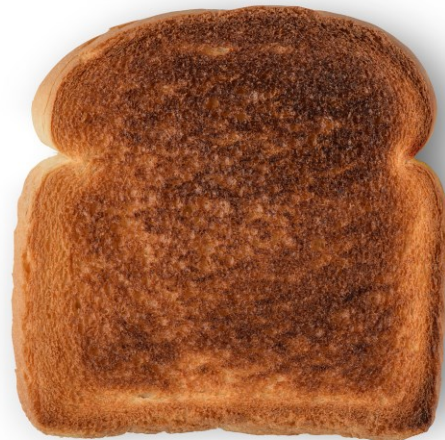
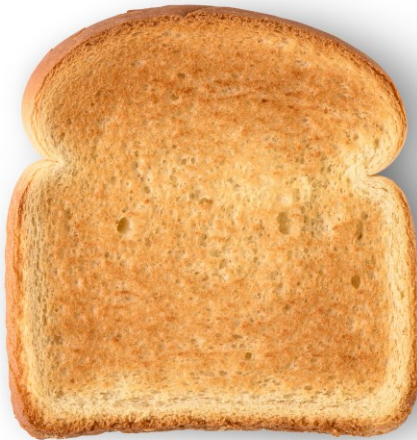
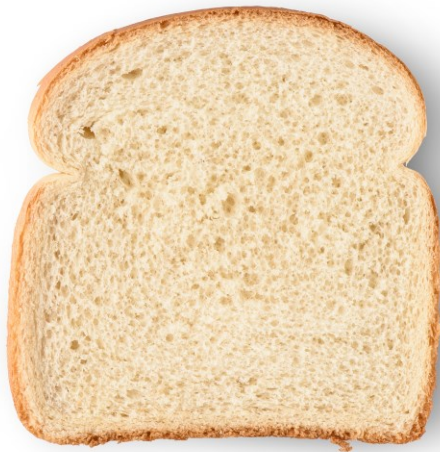
```
<form action="/contacts" method="get" class="toolbar">
  <label for="search">Search Term</label>
  <input id="search" type="search" name="q" value="{ { req
  <button hx-post="/contacts" hx-target="#main">
    Search
  </button>
</form>
```





# DEMO!

---





Recap + Q & A





THANK YOU