



Frontend solutions for Enterprise App

Daniele Morosinotto

#XeOneDay

Evento realizzato grazie
al supporto di



www.xedotnet.org

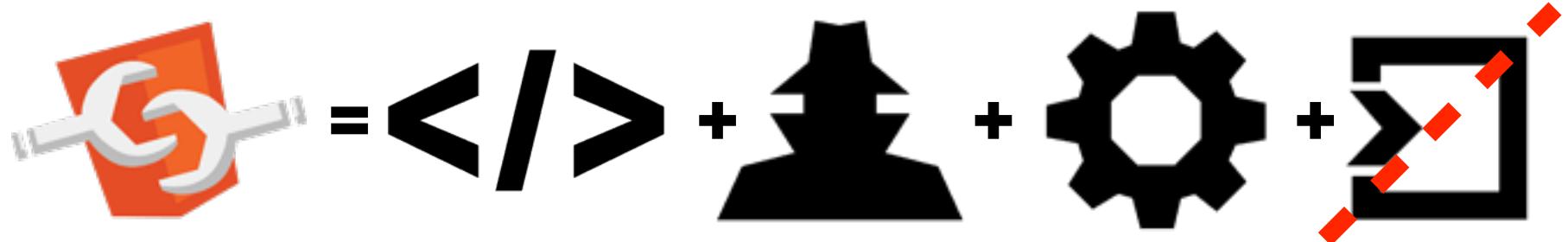
Agenda:



- Intro ai Web Components
- Un pò di storia... “Enterprise”
- Approcci per UI Composition
- DEMO
- Conclusioni

Specifica W3C “Umbrella”

- Custom Elements
- Shadow DOM
- HTML Template
- ~~HTML Import~~



WebComponents Vantaggi:



- E' uno Standard indipendente dai FW JS
- Supporto browser moderni (polyfill IE11+)
- Possibilità di arricchire il “dialetto HTML” con dei <tag-custom> che permettono di isolare e riutilizzare la logica.
- Facilità di utilizzo da “Vanilla JS” e HTML
 - Interfaccia conosciute: DOM event/prop
 - Basta caricare <script src=“tag-def.js”> + usare nel’HTML <tag attr=“x”></tag>
 - Facilità caricamento dinamico (lazyload)



Esempio “code snippet” Web Component

```
class MyComp extends HTMLElement {  
    constructor() { //define HTML or use template+shadowDOM  
        super() // this.attachShadow({mode: "open"})  
        this.innerHTML = "<div>...</div>"  
    }  
    //sample Properties any Object types  
    get myProp { return ... }  
    set myProp(value) {  
        ... //eventually emit Events + data  
        this.dispatchEvent(new CustomEvent("prop-changed",  
            { detail: ... } ))  
    }  
  
    attributeChangedCallback(name, oldValue, newValue) {  
        [...] //handle Attribute change, always String in HTML  
    }  
}  
customElements.define("my-comp", MyComp) //register new TAG
```

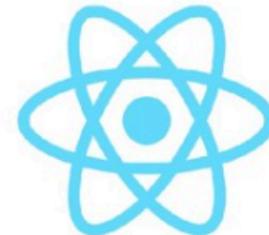
- Tutti i Framework JS moderni supportano l'utilizzo nativo di WebComponent esattamente come gli altri <tag> HTML
- Possibilità di wrappare Componenti specifici dei vari (*👉) FrameworkJS ed esportarli come WebComponent



Angular Elements



Vue.js + Vue CLI



Wrap by hand/
Community Solutions

THE END



</web-component>

<!-- THE END -->

Qui inizia la mia vera sessione 😜



DEFINIZIONE:



“the idea of **micro frontends** is
to extend the **concept of micro services**
to the **frontend** world.”

(micro-frontends.org)



un pò di storia... A yellow emoji of a smiling face wearing black-rimmed glasses.

C'era una volta il “monolite”



the application



the database

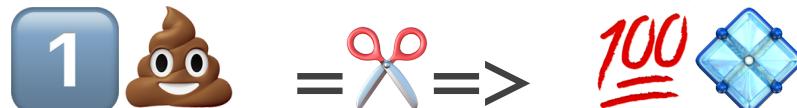
Problemi del “monolite”



- Complessità per numero di funzionalità
- Manutenibilità sul lungo periodo
- Difficoltà di lavoro da parte più Team
- Difficoltà di cambiamenti architettura
 - Problema del “Legacy code” / quale FW scegliere? / Come provare nuovi FW



IDEA: Tagliare



Divisione per “layer”



frontend layer



business layer

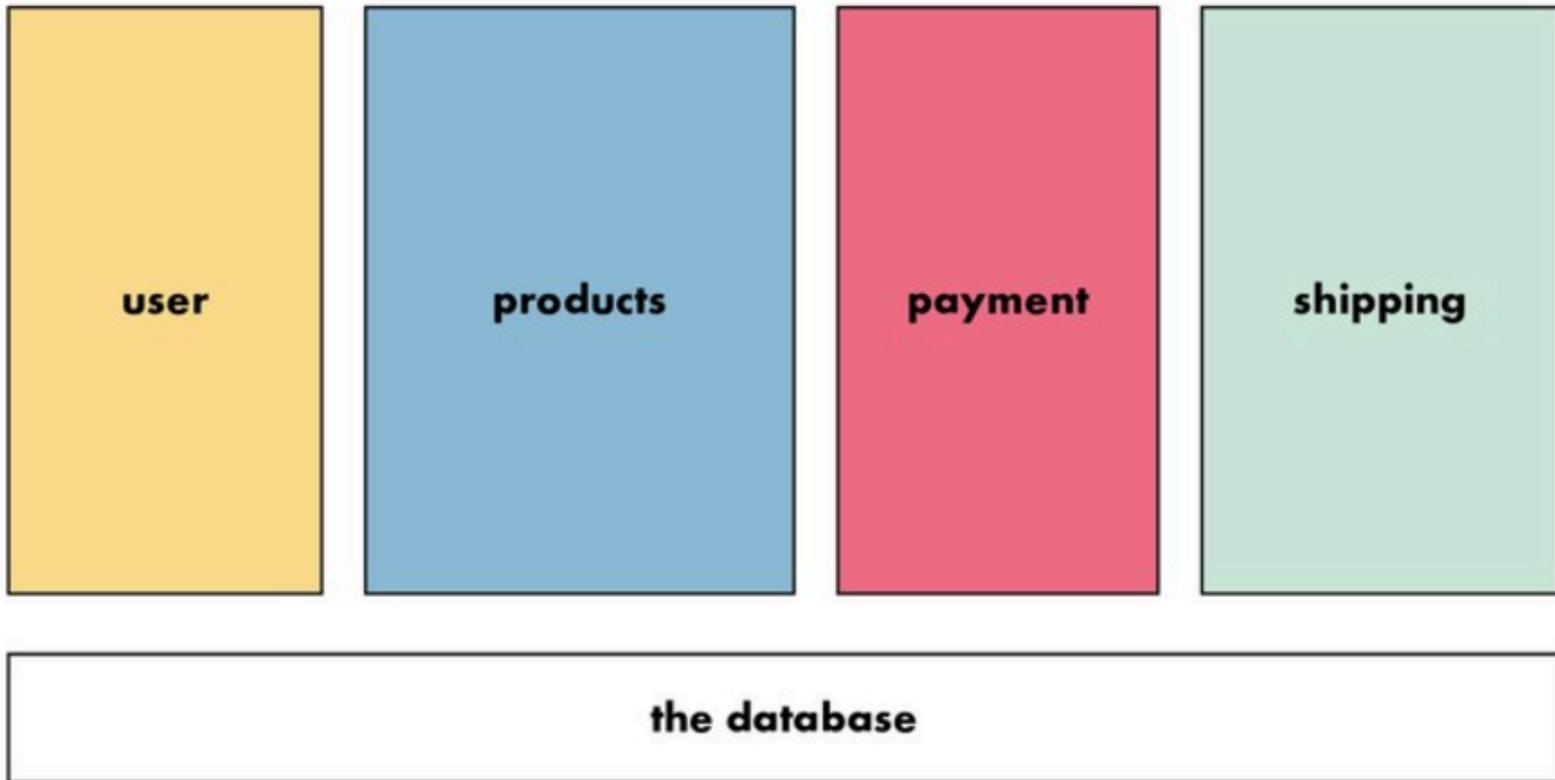


database layer



the database

Approccio a micro-servizi



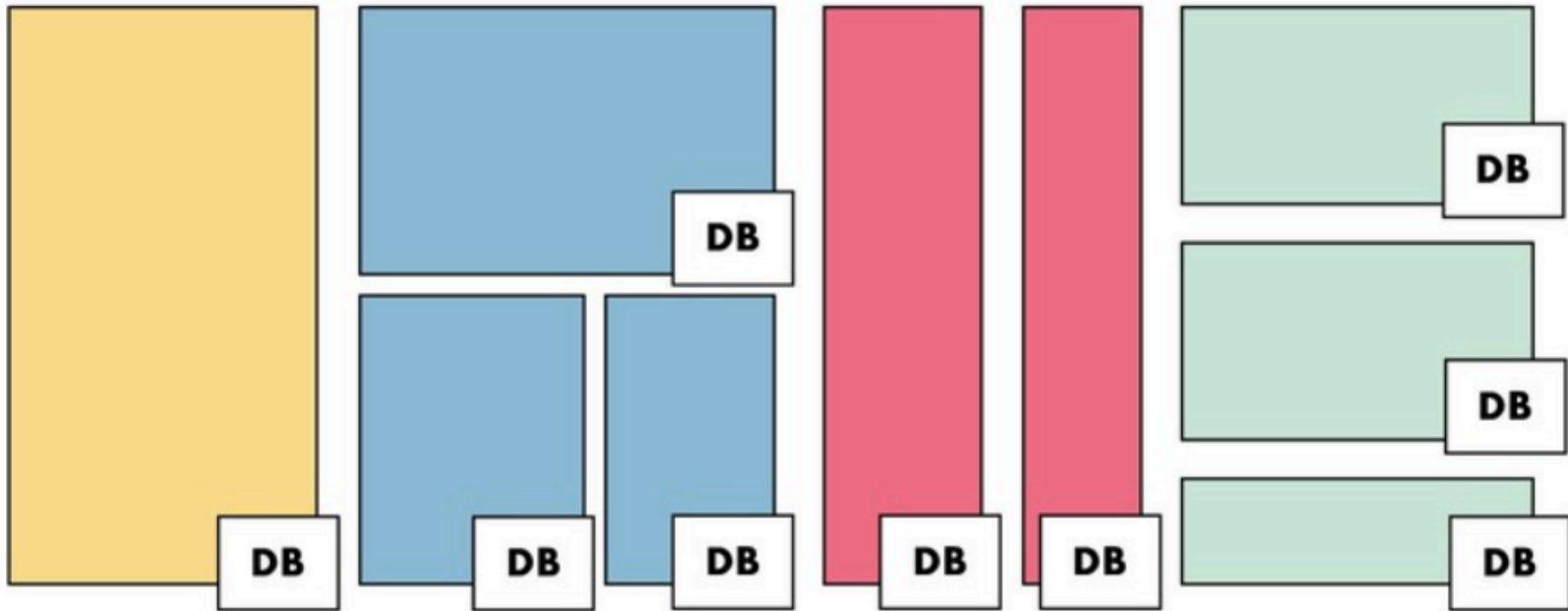


“micro services are small and focused on one thing ... to be **autonomous**. That’s it!”

(Sam Newman)

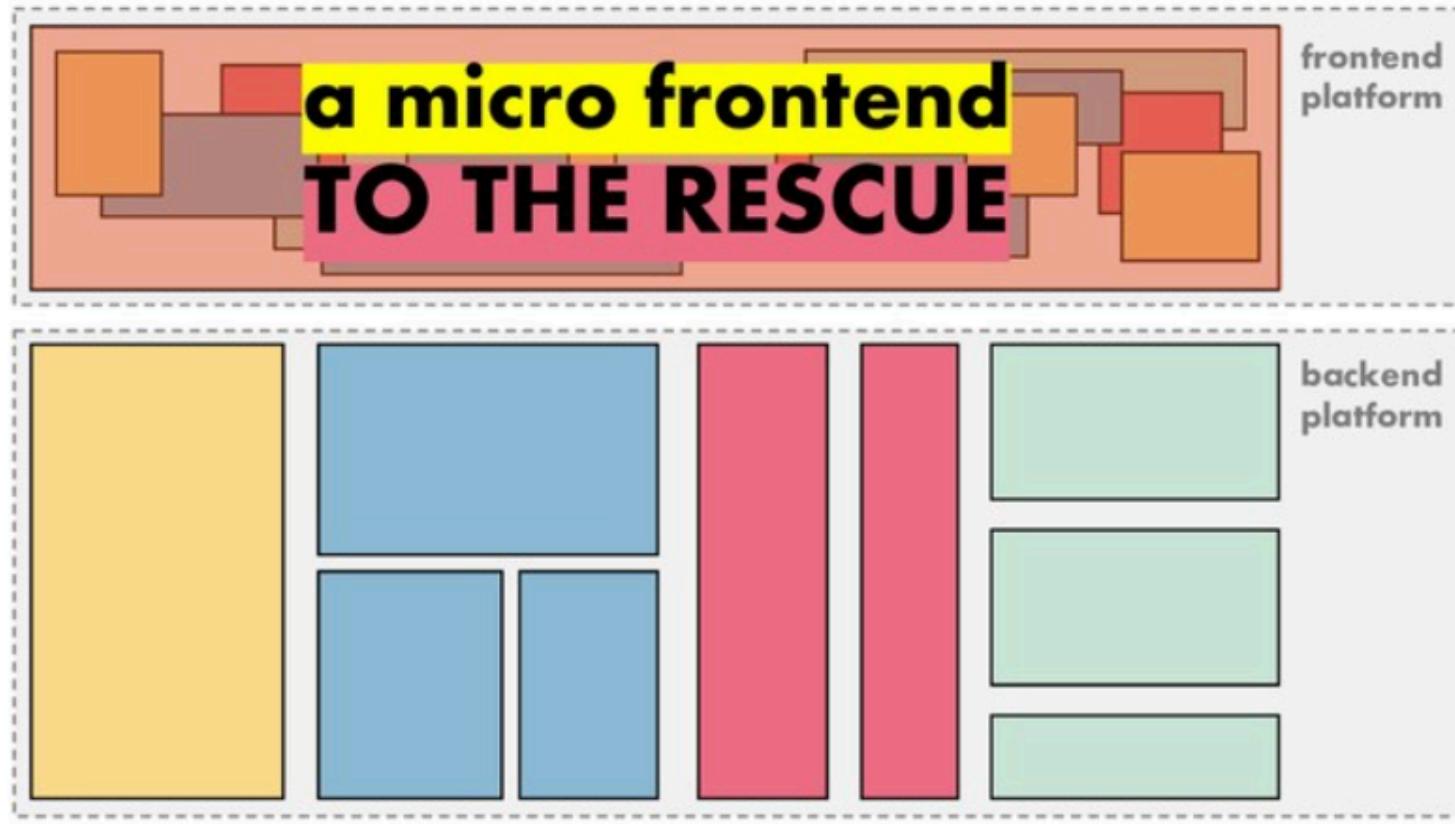
- Più libertà/indipendenza dei Team
- Deploy separato indipendente
- Possibilità di usare/sperimentare differenti tecnologie (e framework)
- “minore complessità” perché si lavora su parti più piccole e dominio ben definito

Svantaggi (CONTRO)



- Dovrebbero essere “Autonomi” ma:
 - Gestione sistema distribuito (comunicazione)
 - Dati distribuiti (Storage/API indipendenti)

Problema frontend “UI Composition”



Complimenti abbiamo di nuovo il nostro “monolite” 💩

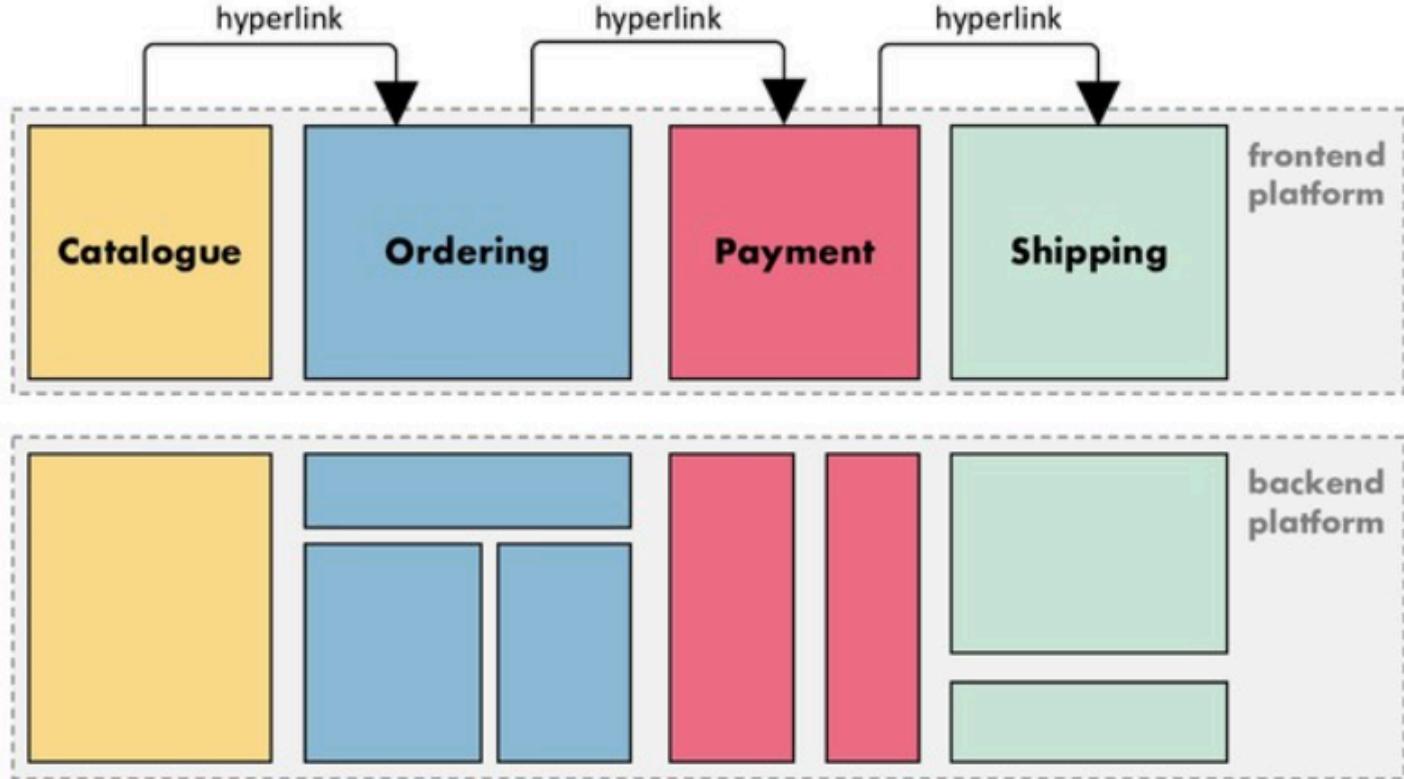
Il frontend non è un dettaglio e va “architetturato” 😎

Approccio #1: Hyperlink



✓ PRO: Semplicità + Sviluppo/Deploy separato

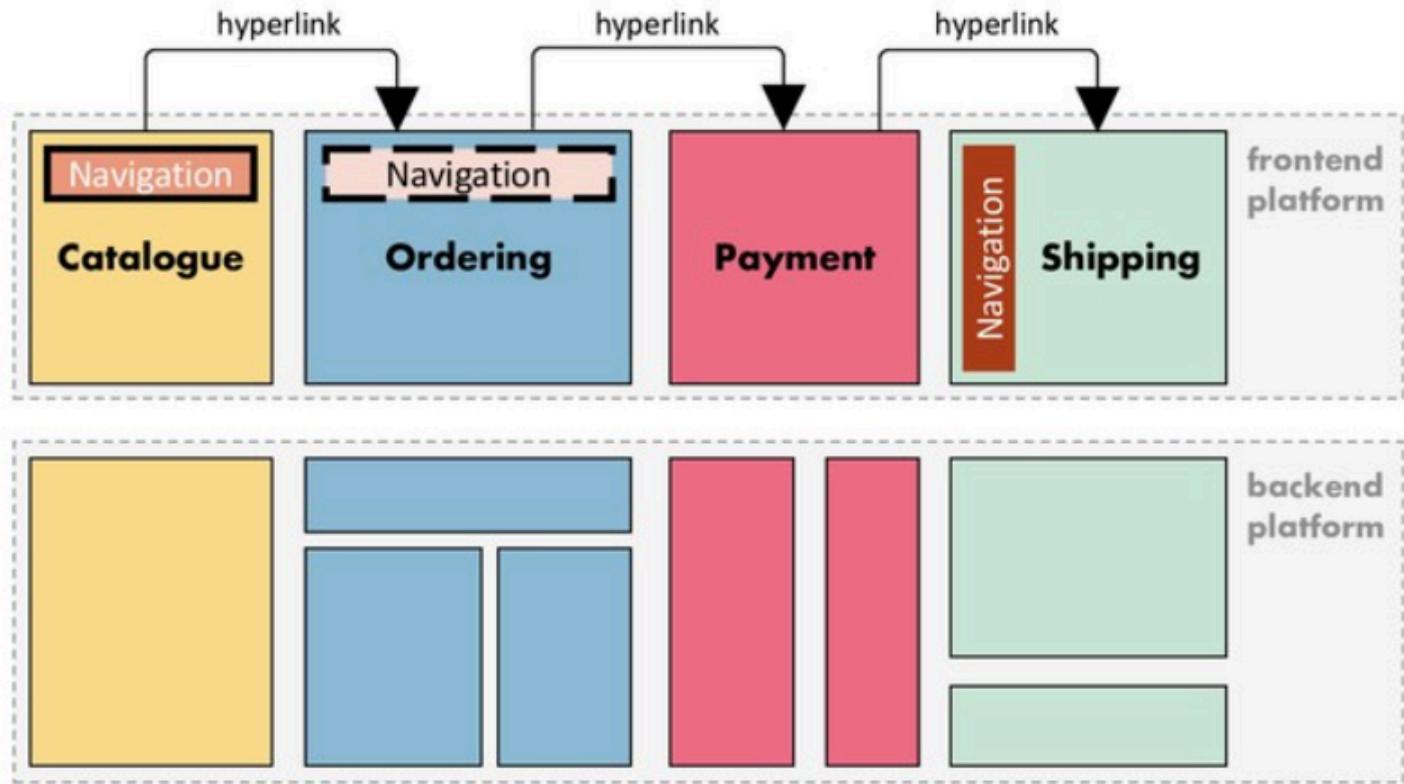
sos CONTRO: Perdita stato (reload pagina/FW)



Approccio #1: Hyperlink



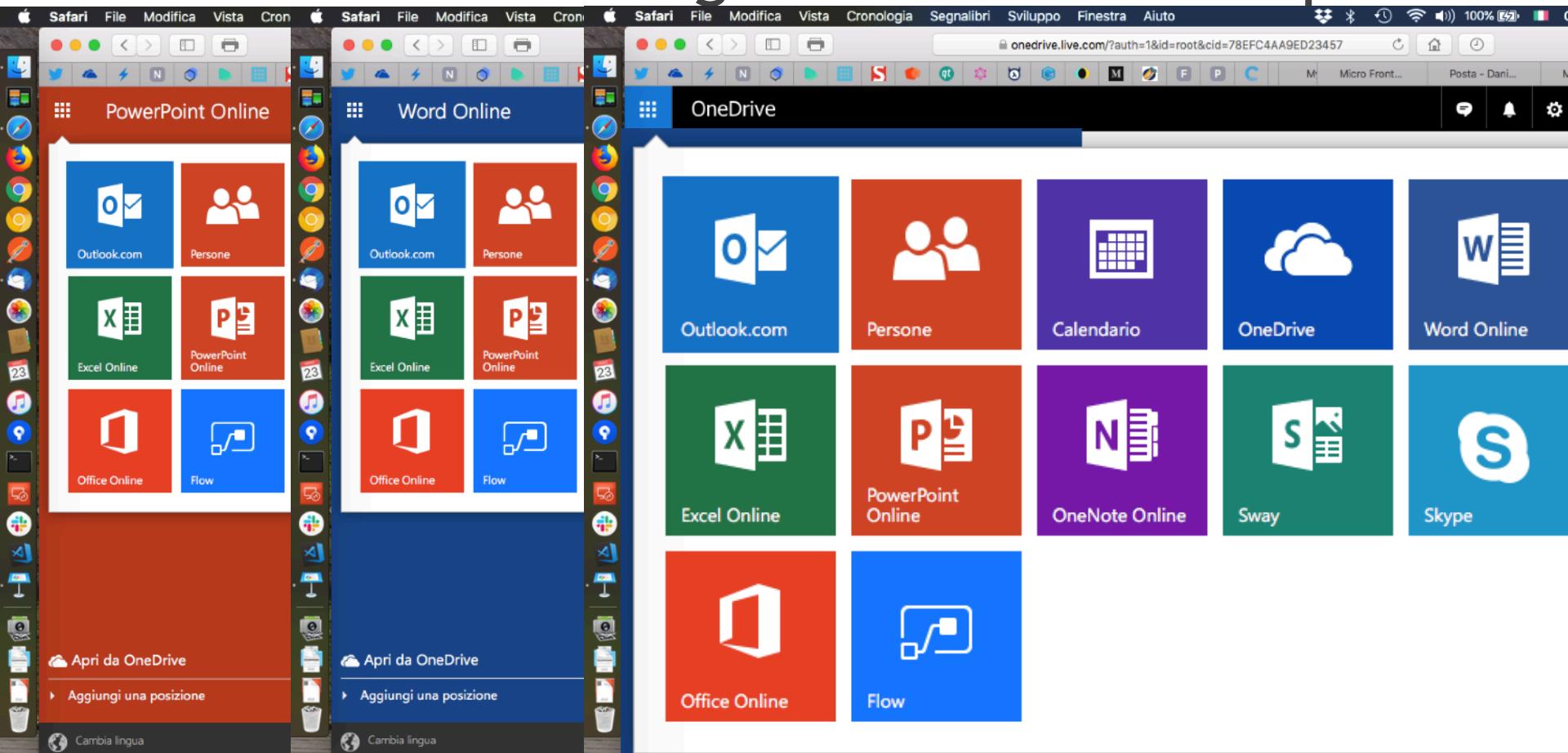
⚠ ATTENZIONE: UI/UX Look&Feel Consistente





Approccio #1: Hyperlink

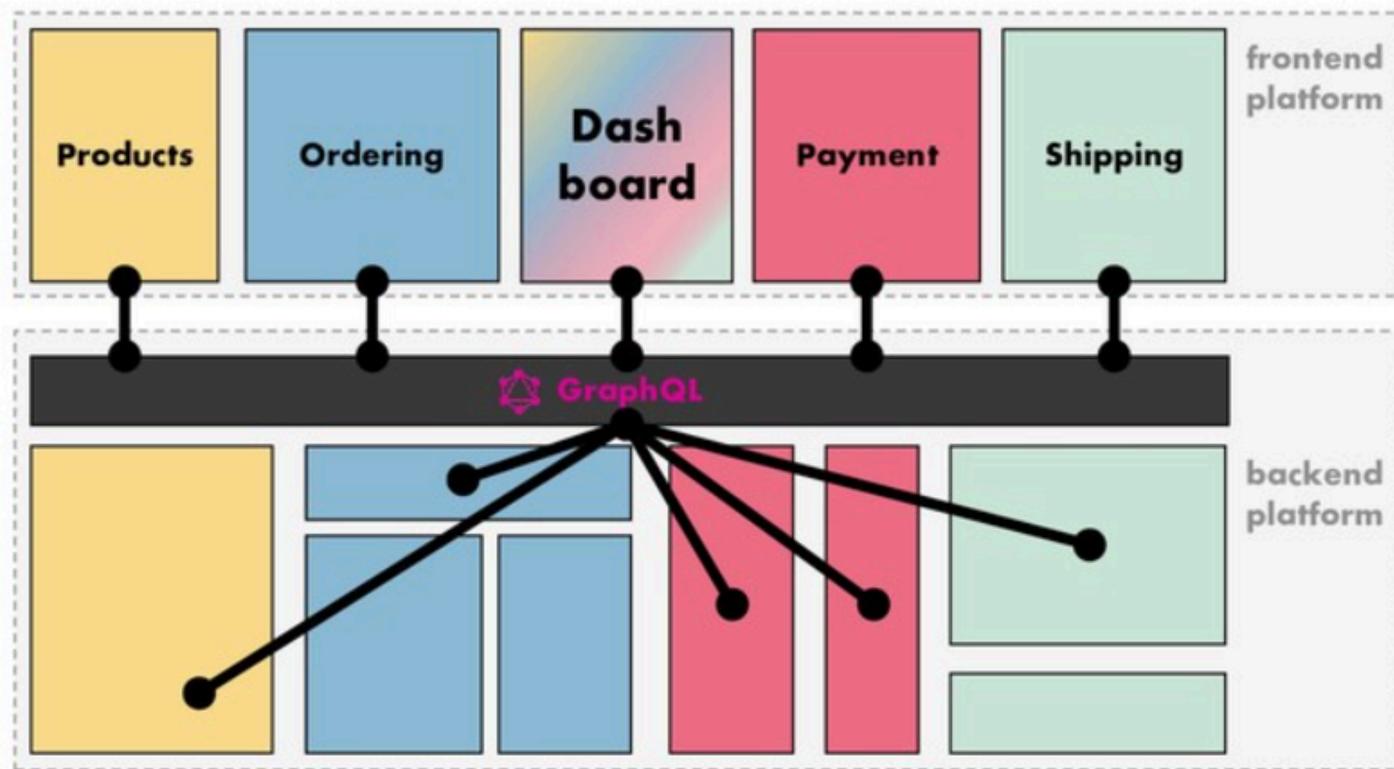
⚠ ATTENZIONE: UI/UX Look&Feel Consistente
💡 IDEA/SOLUZ: Widget UI con WebComponent



Approccio #1: Hyperlink



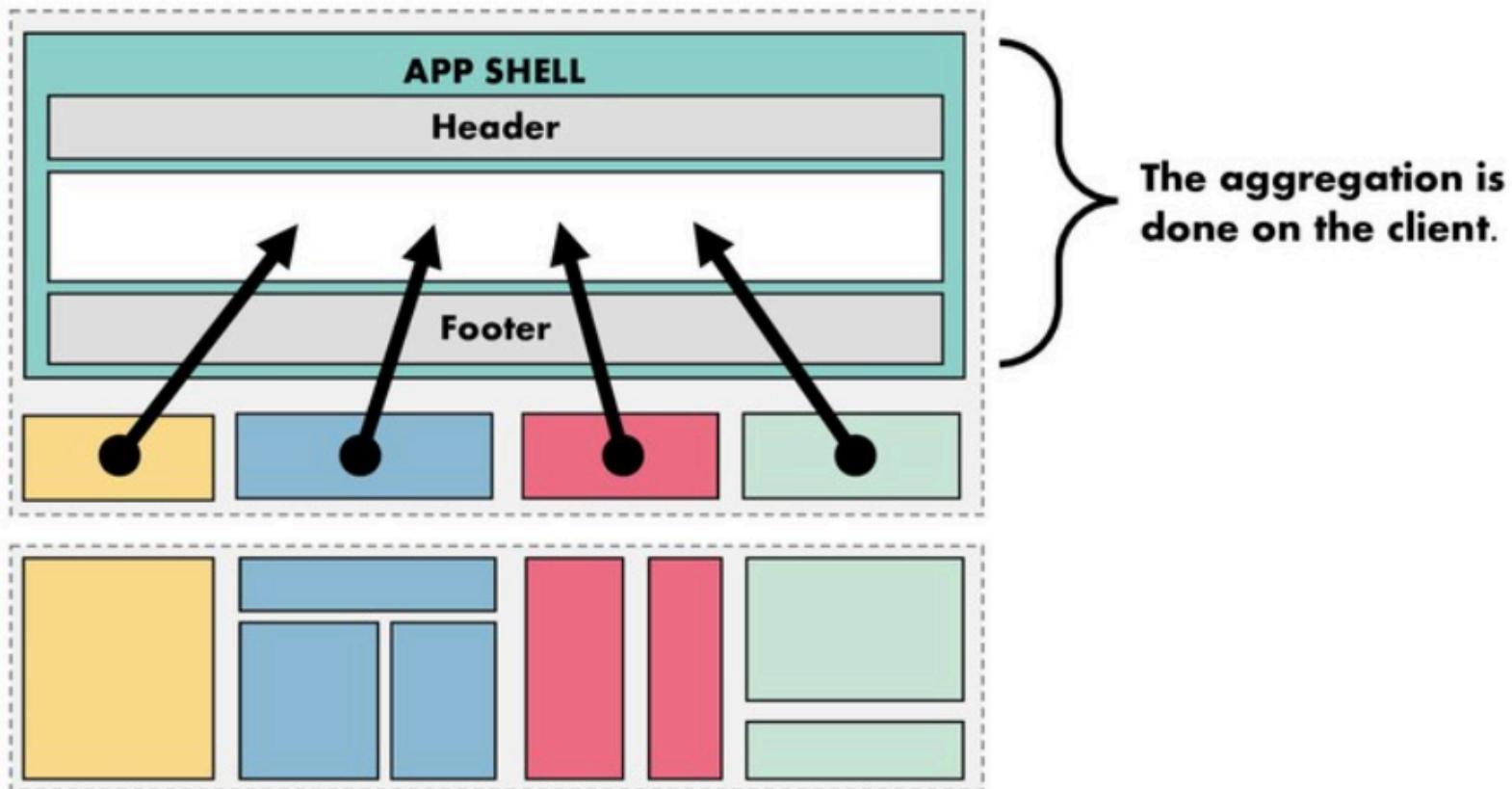
⚠ ATTENZIONE: Pagine “cross”-dominio/slice
💡 IDEA/SOLUZ: Backend for Frontend/GraphQL





Approccio #2: App Shell con

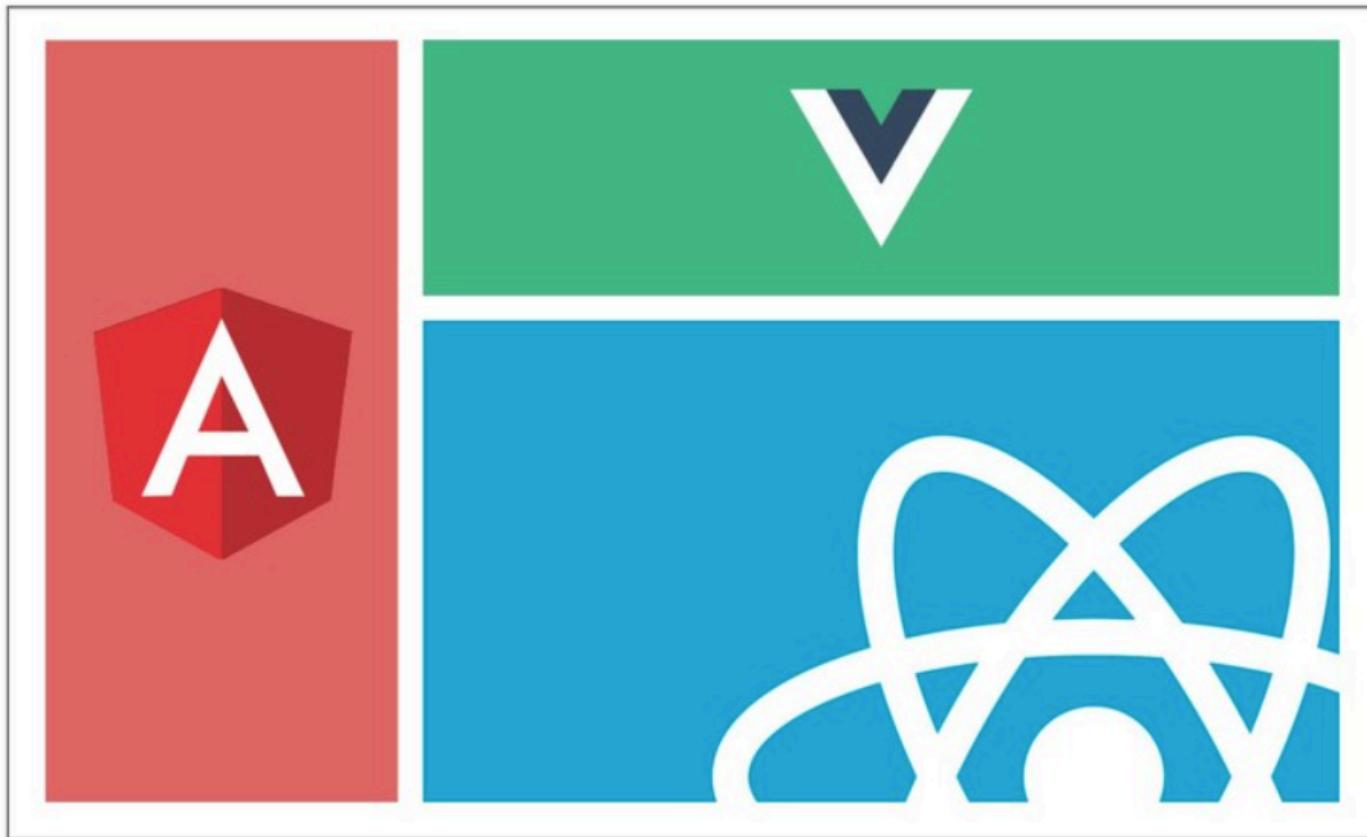
<iframe src="惊吓表情符号">



Approccio #3: App Shell con



💡 IDEA WebComponent = wrapper miniSPA 😎



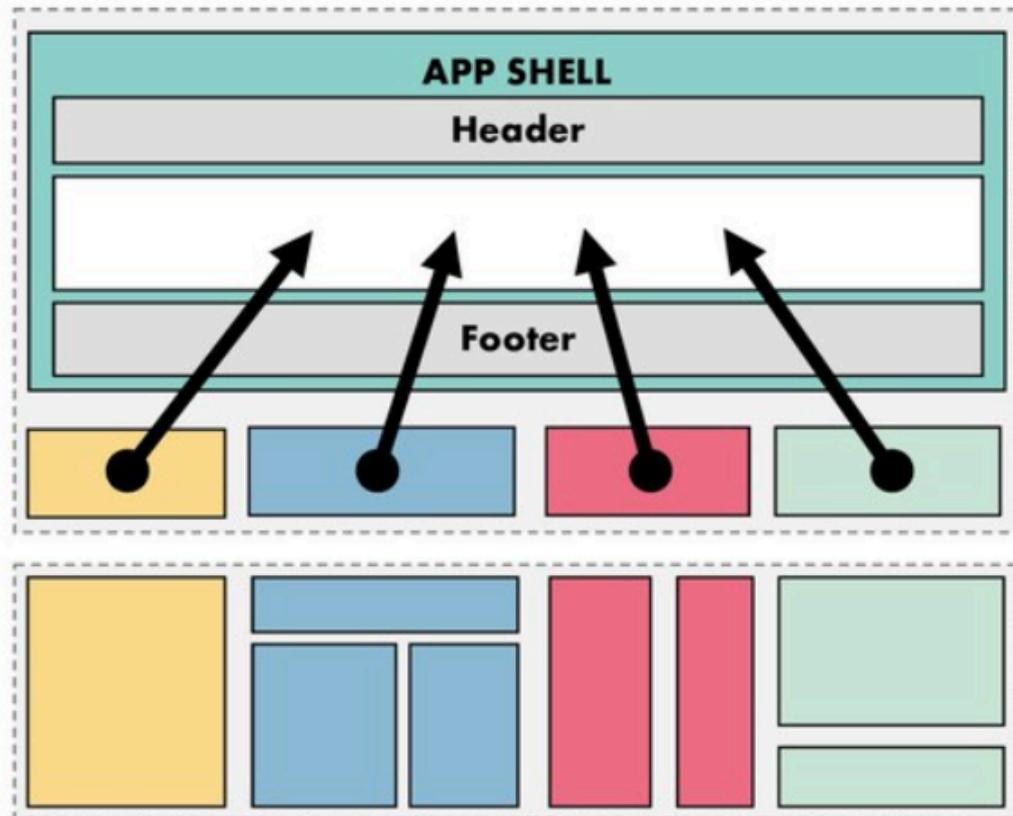
Web Components 

Approccio #3: App Shell

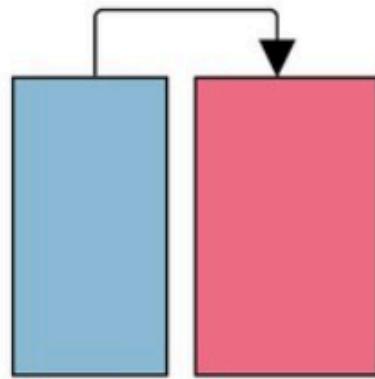


✓ PRO: Abilità diverse tecnologie + UI uniforme

sos CONTRO: Caricamento lento + Interazione? 😕



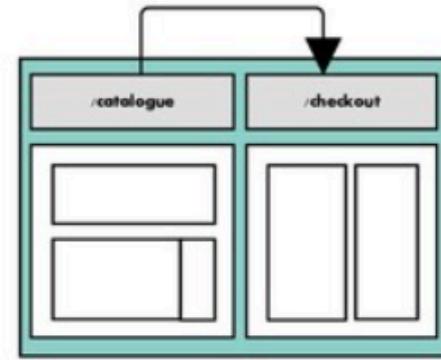
Crescendo di “Interazione” tra i vari Moduli



Hyperlink Integration



App Shell Integration



Metaframework

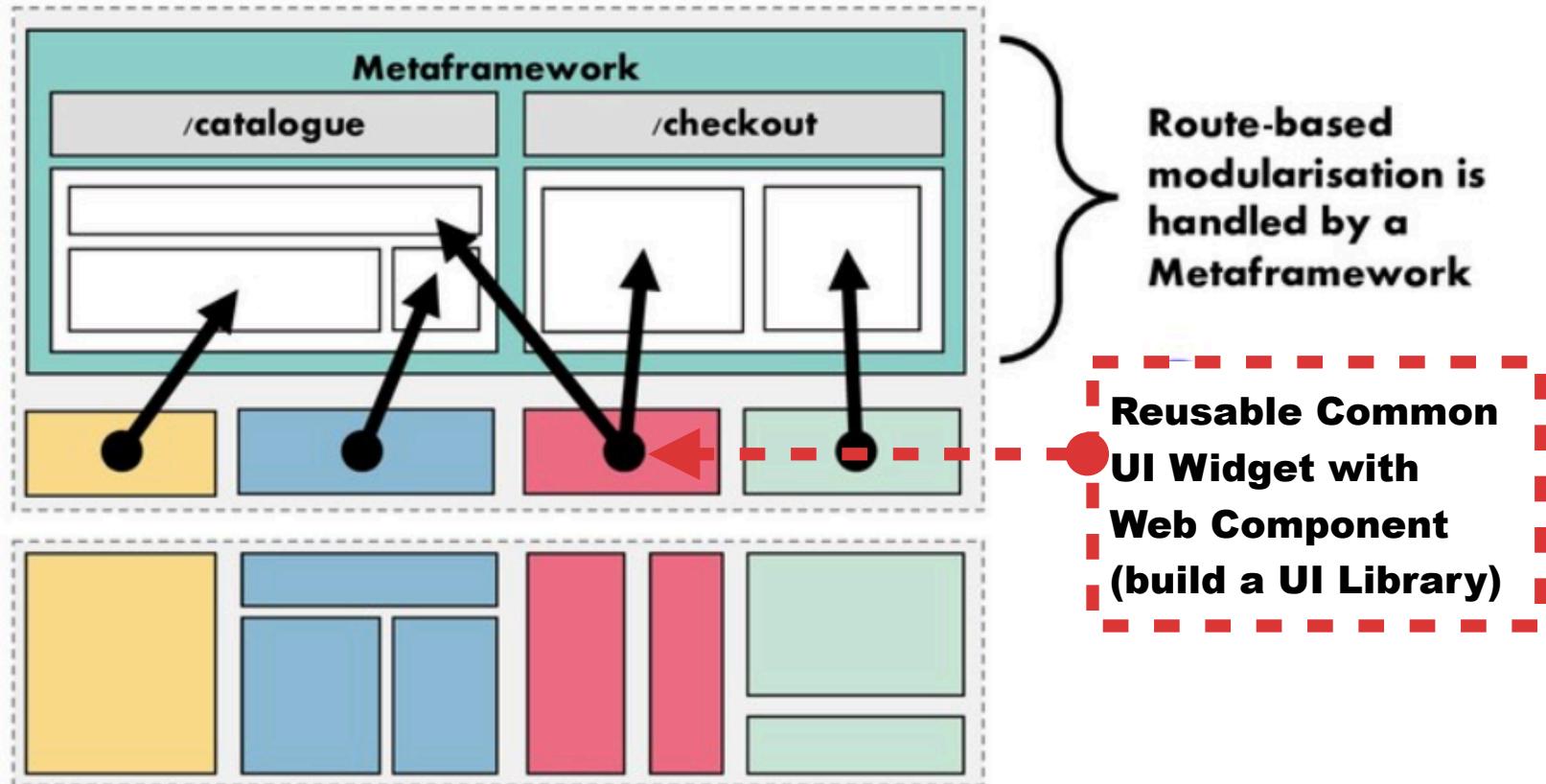


Approccio #4: Macrofrontend + WidgetUI



✓ PRO: Interattività/Comunicazione/Riutilizzo

sos CONTRO: Bundle size (soprattutto se più FW)

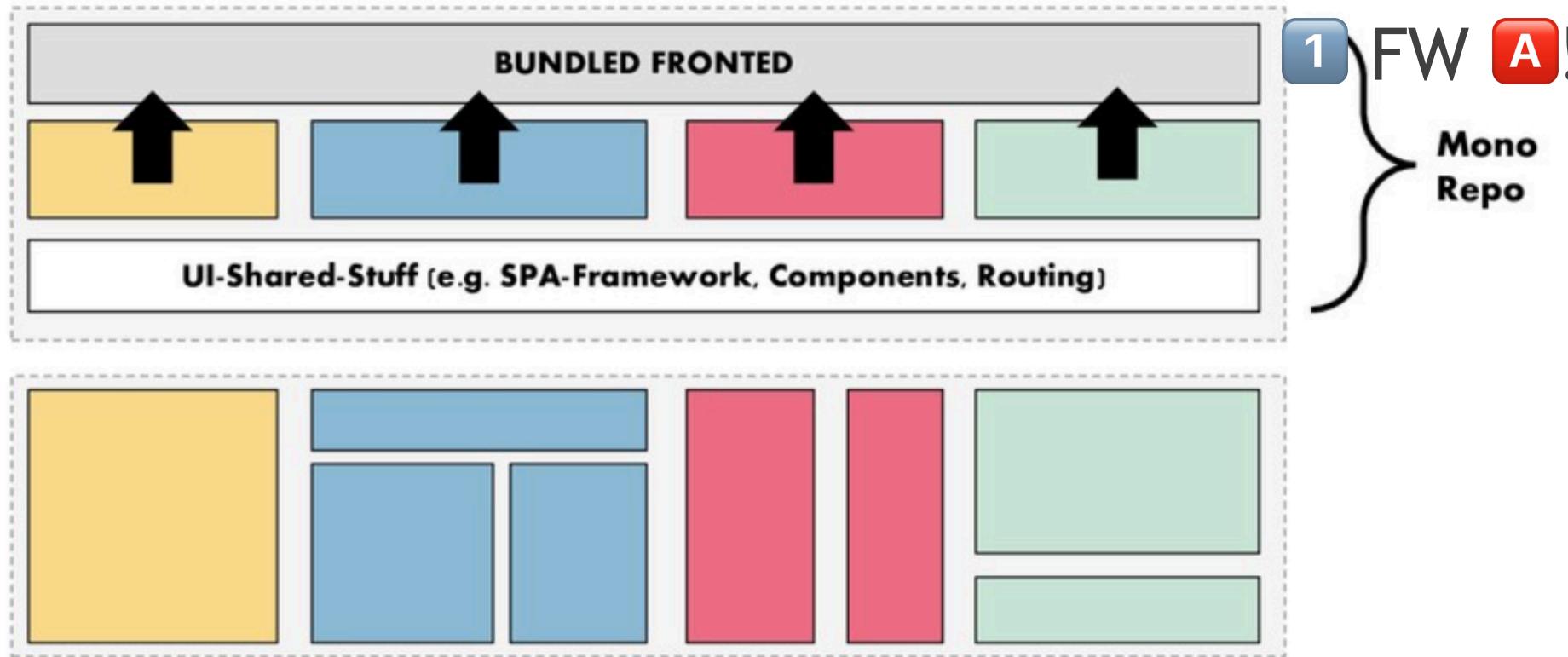


Approccio #5: MonoRepo - Bundle



✓ PRO: Massimizza riutilizzo / Moduli share / Lazy

sos CONTRO: Deploy unico / Build-time / scelta





DEMO

SHELL con WebComponents
utilizzando differenti Framework

```
> cd shell  
> npm i  
> npm start
```

https://github.com/dmorosinotto/XE_OneDay_EnterpriseApp

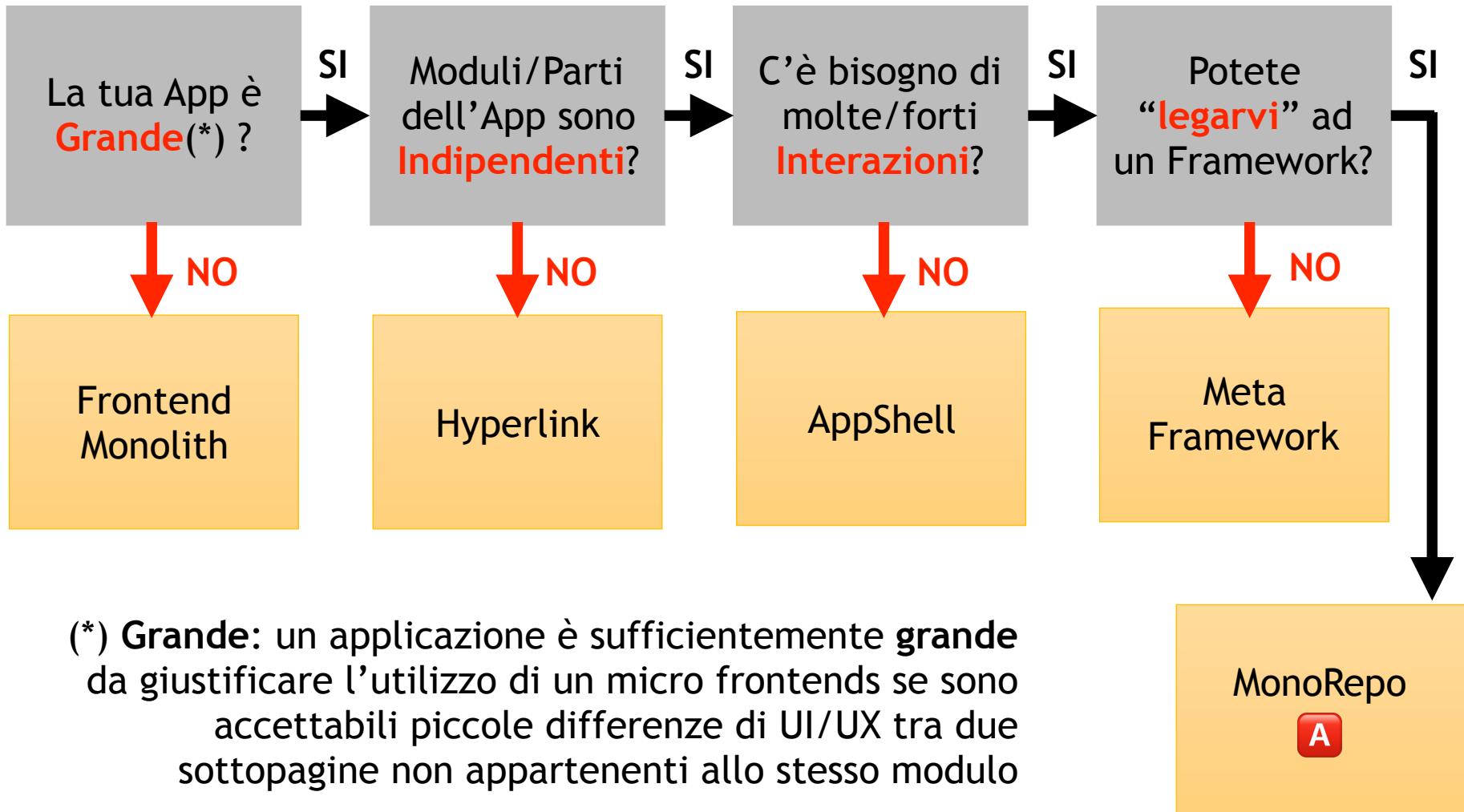
Quale è l'approccio giusto/migliore? 



it
depends ...



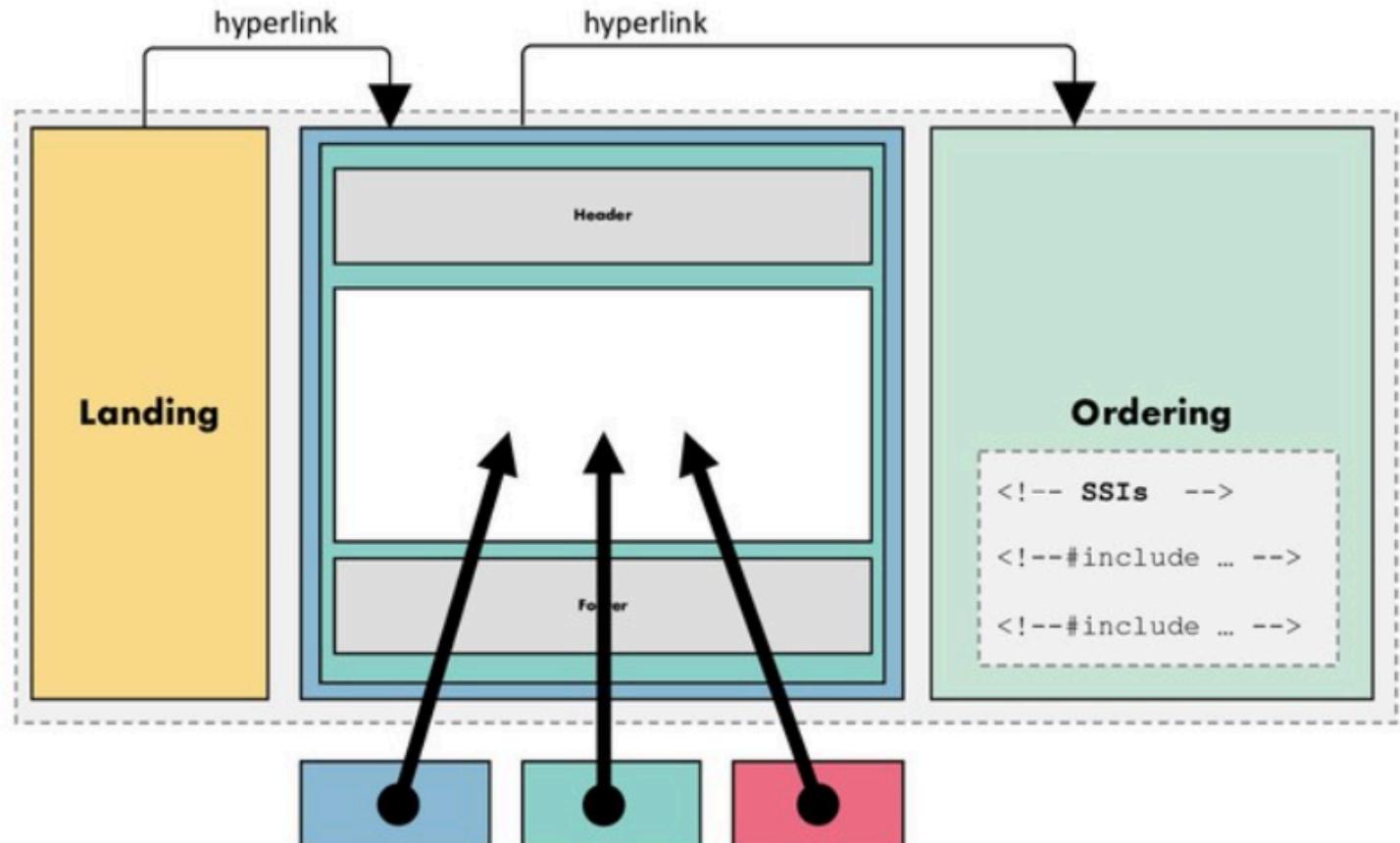
Come scegliere...



E' ammesso "improvvisare" 😊



TIP: Spesso un approccio unico NON risolve tutte le necessità bisogna combinare più soluz.





**“Frontend is ...
NOT an implementation detail,
it is a critical part of your
microservice architecture!”**

KISS: “**think twice** if a frontend monolith does not fit your needs and start with a **monolith first** approach.”

(David Leitner)



Daniele Morosinotto



JavaScript enthusiast



[https://github.com/dmorosinotto/
XE_OneDay_EnterpriseApp](https://github.com/dmorosinotto/XE_OneDay_EnterpriseApp)



d.morosinotto@icloud.com



[@dmorosinotto](https://twitter.com/dmorosinotto)

Look mom

I'm here 😊

Thanks to *you!*