Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Triennale in Ingegneria Informatica

Microfrontends

Candidato
Matteo Bavecchi

Relatori
Prof. Romano Fantacci
Prof. Giuseppe Pecorella

Correlatore
Dott. Andrea Rizzo

Anno Accademico 2020/2021

Indice

ringraziamenti

Grazie a tutti

Introduzione

Organizzare il lavoro per lo sviluppo di applicazioni web di grandi dimensioni non è per niente banale, ed è molto interessante capire come suddividere responsabilità tra i vari team che contribuiscono alla sua realizzazione.

L'approccio più diffuso per affrontare il problema è quello di suddividere le persone per competenze (ovvero in modo *orizzontale*), creando team che mettono in comune figure con abilità dello stesso ambito. Il classico approccio monolitico infatti prevede due suddivisioni, la parte *frontend*, che si occupa dell'esperienza utente e delle interfacce e la parte *backend*, che invece gestisce server e basi di dati. Ad esempio in un sito e-commerce possiamo trovare un team che si occupa della parte frontend, uno che cura i servizi di pagamento e uno che segue la parte backend.

Quando il progetto aumenta di complessità, si sente la necessità di suddividere il lavoro in sotto-progetti, e l'approccio orizzontale potrebbe non essere la scelta migliore, in quanto rallenta l'introduzione di nuove funzionalità.

Possiamo allora pensare di assegnare ad ogni team una parte del progetto, i quali dovranno portarlo al termine interamente. Ogni team avrà bisogno quindi di competenze eterogenee al loro interno.

Abbiamo così il superamento dell'approccio monolitico, e possiamo parlare di microfrontend.

Questo già avviene analogamente nello sviluppo backend, dove l'appli-

cazione viene vista come un insieme di *microservizi* disaccoppiati e con granularità fine.

Capitolo 1

Microfrontend

L'obiettivo della tecnologia microfrontend è quello di superare l'approccio monolitico, che vede lo sviluppo di applicazioni web suddiviso in due teams: backend e frontend.

Si fa questo vedendo un'applicazione web come un insieme di elementi, chiamati fragment o microfrontend, molto disaccoppiati tra di loro e con la più bassa granuralità, ovvero con la funzionalità più minimale possibile.

Ogni microfrontend viene sviluppato da un team, che potrà lavorare con autonomia. Essendo i singoli microfrontend autonomi, questi possono funzionare anche se estratti dalla applicazione web che li contiene, e il malfunzionamento di un singolo microfrontend non compromette la stabilità degli altri.

Vantaggi

- **Ottimizzare lo sviluppo di funzionalità:** Nell'approccio orizzontale quando si vuole sviluppare una nuova funzionalità è necessario far convergere il lavoro di più team. Con microfrontend, tutte le perso-

ne coinvolte nell'implementazione di una nuova funzionalità sono nello stesso team, rendendo il lavoro più veloce ed efficiente.

- **Abbandono del frontend monolitico:** Con microfrontend le applicazioni, incluse il frontend, si dividono in sistemi verticali più piccoli. Ogni team controlla la sua piccola parte di frontend e di backend.

- **Adottare diverse tecnologie:** Strumenti di sviluppo e framework evolvono continuamente. Ogni team deve essere in grado di scegliere le proprie tecnologie autonomamente. Ci sono alcune grandi aziende come Github, che hanno impiegato molto tempo per eliminare alcune dipendenze ormai obsolete dal loro codice (nel caso di Github si trattava di una versione di JQuery). Con l'approccio microfrontend questi cambiamenti sono più rapidi e possono essere fatti modularmente.

- **Indipendenza:** I progetti di ogni team sono autonomi, ovvero non hanno dipendenze condivise tra di loro. Questo come già detto precedentemente porta ad una grande autonomia.

  L'indipendenza però porta sicuramente a costi aggiuntivi. Si potrebbe pensare che sia più semplice quindi di sviluppare un unico progetto, e assegnare parti di questo a team diversi. Il problema sta nel fatto che la comunicazione tra i vari team è costosa e porta molti ritardi. In alcuni casi quindi è preferibile introdurre ridondanza nel codice dei vari team a favore di più autonomia e velocità di implementazione.

Svantaggi

- **Ridondanza:** In informatica si è addestrati a ridurre al minimo ridondanze. Anche nel caso dello sviluppo frontend ci sono degli episodi nei

quali la ridondanza può essere molto costosa: come ad esempio quando viene trovato un bug in una libreria e questo viene risolto da un team, il fix dovrà essere comunicato agli altri e questi dovranno provvedere a risolverlo autonomamente. Oppure quando si rende un processo più veloce, anche in questo caso va comunicato agli altri team e questi dovranno apprendere la scoperta. Si adotta quindi un approccio microfrontend quando i costi associati a ridondanze sono inferiori agli impatti negativi a uno sviluppo frontend monolitico, che porta a forti dipendenze tra team.

- **Inconsistenza:** Potrebbero essere presenti delle basi di dati, necessarie all'applicazione web, che vengono lette e scritte da più microfrontends. Per garantire la proprietà di indipendenza tra team è necessario replicare il database per tutti i progetti. Tutte queste copie devono però essere sincronizzate tra loro regolarmente per mantenere la consistenza e la coerenza dei dati. Questo introduce dei ritardi, che potrebbero penalizzare l'esperienza utente.

- **Eterogeneità:** Potrebbe essere controverso avere la libertà di utilizzare tecnologie diverse tra i vati team. Ovviamente se ogni team usa una diversa tecnologia lo scambio di pareri o di competenze tra team diventa più difficile.

Capitolo 2

Collegare i microfrontends

Per far convivere i progetti realizzati dai vari team esistono varie metodologie. Questo presuppone che i team, nella loro autonomia, debbano comunque scambiarsi un minimo di informazioni e di vincoli che servono alla corretta integrazione delle app web, chiamiamo questo insieme di dati *contratto tra team*.

Collegamento tra pagine con links

La soluzione più semplice è quella del collegamento tra pagine con link URL: in questo modo il contratto consiste negli indirizzi delle pagine, che i team dovranno scambiarsi tra di loro. Questa soluzione è quella che rende i team il più autonomi e disaccoppiati possibile, inoltre si ha grande robustezza, in quanto se un progetto si corrompe, questo non influenza minimamente gli altri, che possono anche essere detenuti in server diversi.

Lo svantaggio è che in una pagina web è possibile contenere informazioni provenienti da un solo team.

Questa soluzione viene usata quando è richiesta una forte robustezza, oppure quando si deve implementare i microfrontends in una app legacy già esistente.

Composizione tramite iframes

L' iframe è un elemento HTML che permette di incorporare un'altra pagina HTML all'interno di quella corrente. [**?**]

Questo elemento può essere usato per comporre fragments provenienti da teams differenti, visualizzandoli su un'unica pagina web. Un team si impegnerà a realizzare la pagina ospitante e dovrà far presente agli altri attori lo spazio riservato ai loro fragments per evitare problemi di visualizzazione, questo dato, oltre che agli indirizzi URL delle varie pagine, farà parte del contratto tra i team.

Un grande svantaggio dell'uso degli iframes è la loro incompatibilità con i motori di ricerca. Infatti le informazioni che vediamo nella pagina non sono in unico file HTML, di conseguenza il motore di ricerca non profila le informazioni contenute negli iframes.

Composizione con Ajax

Un modo per superare il problema dei motori di ricerca ai quali sono affetti gli iframes è quello di caricare i file HTML con Ajax.

La pagina ospitante, detta *wrapper* ha il compito di caricare nel proprio DOM (Document Object Model) i fragments dei vari team, con l'ausilio di un codice javascript, che utilizza la funzione fetch().

Il problema principale però è che l'Ajax request è asincrona, questo porta a dei ritardi nel caricamento completo della pagina.

Utilizzo del Routing Server-side

Il routing è un elemento fondamentale dell'architettura microfrontend. L'elemento che stiamo introducendo viene chiamato frontend proxy, cioè un web server che effettua routing, che intercetta le richieste con un certo percorso e le instrada al giusto fragment. In questo modo anche se i progetti dei vari team risiedono in spazi diversi, l'utente vedrà un URL omogeneo e non si accorgerà delle origini dei fragment. Il funzionamento consiste in questi passaggi:

- Il client apre l' URL "/foo/bar". La richiesta raggiunge il frontend proxy
- Il frontend proxy confronta il path "/foo/bar/"8 con la propria routing table, con corrispondenza in prossimità della regola i
- Il frontend proxy passa la richiesta al fragment associato
- Il fragment genera una risposta e la ritorna al frontdend proxy
- La risposta arriva infine al client

Composizione Server-side

Si suppone di lavorare ad un progetto microfrontend incentrato principalmente su contenuti e informazioni, senza particolari applicativi logici al suo interno. Un esempio di sito incentrato sulle informazioni(anche detto *content-centric*) è Wikipedia, le sue caratteristiche principali sono:

1. Tempi brevi di caricamento delle pagine
2. Ottimizzazione per motori di ricerca
3. Assenza di particolari contenuto logici basati sull'interazione con l'utente

Si introduce quindi la **composizione server-side**, che rispetta le richieste di un sito content-centric come Wikipedia.

La composizione dei fragment viene eseguita da un servizio che risiede nel server web, il client quindi riceve la pagina completa.

Come nel caso degli iframes, i team che producono i fragments devono fornire al team che detiene la pagina ospitante l'URL del loro codice di markup HTML.

Il team della pagina ospitante userà delle direttive per richiedere al server di aggiungere il codice di markup degli altri team in un preciso posto della schermata.

Un esempio di servizio di composizione è la funzione **SSI** (Server-Side Includes) di **Nginx**:

Server-Side Includes

Nginx contiene il modulo *ngx-http-ssi-module* che processa i comandi SSI. I comandi SSI sono istruzioni inserite nelle pagine HTML della pagina

ospitante. Un'istruzione SSI appare nella seguente sintassi:

```
<!--#include virtual="url/da/includere -->
```

Avviene quindi questo:

1. Il client effettua la richiesta della pagina
2. Il webserver accetta la richiesta e verifica la presenza di eventuali direttive SSI
3. Le direttive trovate vengono sostituite con il codice di markup reperibile all'url dell'attributo "virtual"
4. Il client riceve la pagina completa

Si noti che le prestazioni di composizione sono nettamente aumentate rispetto alla composizione con iframes, in quanto il client riceve la pagina già completa, effettuando un solo handshake HTML, indipendentemente dal numero di fragment ospitati nella pagina. Inoltre i motori di ricerca possono profilare anche il contenuto dei fragments

Modalità di caricamento dei fragments

Al contrario della soluzione con iframes, lo scorretto caricamento di un fragment può rallentare o addirittura bloccare il caricamento dell'intera pagina. Il webserver infatti prima di restituire la pagina aspetta di avere tutti i suoi componenti. La proprietà di Nginx chiamata *proxy read timeout* permette di fissare un tempo massimo per il quale il webserver può aspettare un fragment.

Nel caso in cui il webserver non riesce a caricare un fragment esiste il comando SSI **stub**. Si utilizza la direttiva *block* che sostiuirà il fragment

caricato senza successo con del codice HTML di riserva. Il nuovo *include* contiene il nuovo attributo stub, che ha il riferimento al block corrispondente, dovesse non andare a buon fine la richiesta. TODO: METTI SNIPPET STUB

Alternative a SSI

Zalando Tailor

Podium

Composizione Client-side

Comunicazione tra fragments

Diretta

Tramite il padre

Event-bus

Application Shell

Routing a due livelli

Universal Rendering

Capitolo 3

Conclusioni

...