Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Integrazione e migrazione di applicazioni Web-based attraverso l'uso di micro-frontend: un caso di studio aziendale

Elaborato in:
APPLICAZIONI E SERVIZI WEB

Relatore:
Prof.ssa
Silvia Mirri

Presentata da:
Luca Maestri

Sessione III
Anno Accademico 2020-2021

A tutti i nonni.

Introduzione

A partire dalla nascita del Web e dalla sua diffusione abbiamo assistito ad una costante evoluzione dei contenuti e dei servizi offerti tramite esso, oggi i requisiti e le aspettative legati alle applicazioni Web-based sono sempre maggiori, infatti gli utenti desiderano che l'accesso ai contenuti forniti sia sempre più rapido, le interfacce presentate siano facili da usare ed immediate oltre che estremamente reattive, è richiesto che i servizi forniti siano accessibili da una vasta gamma di dispositivi con caratteristiche diverse e soprattutto che siano sempre più ricchi di nuove funzionalità.
Attualmente la tendenza è quella di realizzare applicazioni web in grado di offrire svariati servizi, con l'obiettivo di soddisfare tutte le aspettative degli utenti, questo sta portando alla nascita di sistemi sempre più complessi, dove team composti da un numero considerevole di persone si trovano alle prese con varie sfide da superare.
Per rispondere a questo genere di necessità si è diffusa in particolar modo la tendenza ad affidarsi ad architetture basate su microservizi, ciò ha permesso di ottenere maggiore flessibilità e resilienza, oltre che importanti vantaggi in termini di scalabilità. Usare questo genere di approccio consente di aggirare le limitazioni e gli svantaggi caratteristi dei grandi backend monolitici, largamente diffusi in precedenza, mentre lato server sono stati quindi raggiunti importanti traguardi in questo senso, lato frontend si è diffusa la tendenza a realizzare delle Single Page Application e più in generale ad adottare strategie monolitiche, complesse e difficilmente manutenibili.
Il termine micro-frontend è apparso per la prima volta all'interno di Thought-

Works Technology Radar alla fine del 2016 [1], in questo approccio l'applicazione web viene suddivisa nelle sue funzionalità peculiari e ciascuna di esse è di proprietà, dal frontend al backend, di un team diverso, ciò garantisce che vengano sviluppate, testate e distribuite in maniera indipendente dalle altre.

L'obiettivo del lavoro presentato, è di illustrare le potenzialità fornite da una architettura a micro-frontend attraverso il supporto di un caso di studio aziendale, in particolare all'interno del Gruppo Maggioli, il quale riguarda l'esigenza di attuare un processo di trasformazione che porti all'integrazione e alla migrazione di diversi applicativi caratterizzati da funzionalità spesso comuni, sia vista la necessità di ottimizzare le risorse investite all'interno dell'organizzazione, sia per il bisogno di applicare un processo di rinnovamento a fronte dell'approssimarsi dello stato di obsolescenza di certe tecnologie in uso ma anche per sfruttare le potenzialità messe a disposizione da un settore in continua mutazione.
Il tutto attuando una evoluzione che dovrà tenere in considerazione le varie difficoltà organizzative, comuni ad aziende di medio e grande dimensioni, caratterizzate da più team di lavoro dislocati sul territorio. Un altro punto fondamentale sarà l'abbandono dei precedenti applicativi, adottando un approccio basato sull'idea di generare valore in maniera incrementale e costante, fondando il nuovo sistema sulle basi di quello attuale, in modo da non dover sospendere il rilascio di nuove funzionalità verso i propri clienti per riuscire, solo dopo un lungo tempo, a fornire un'unica nuova soluzione software.

La tesi è strutturata come segue:

- *Capitolo 1*: in principio verrà presentata una panoramica sullo stato dell'arte delle architetture basate su micro-frontend, ripercorrendo l'evoluzione storica che ha portato il Web sino alla sua forma attuale, verranno quindi definiti i principi di questa architettura in relazione anche con un approccio a microservizi, partendo dalle motivazioni che

hanno spinto verso la ricerca di alternative rispetto ad un classico sistema monolitico. Al termine, saranno analizzati i benefici e i vantaggi della soluzione mentre nel seguito verranno inquadrati i principali temi da trattare prima di intraprendere questa strada, sarà fornito uno sguardo che prescinde da specifici framework e si focalizza sui ragionamenti e sulle considerazioni da effettuare in base al proprio specifico contesto di utilizzo.

- *Capitolo 2*: in questo capitolo verrà presentato il contesto aziendale del Gruppo Maggioli, illustrando le principali tappe storiche che hanno portato l'organizzazione sino al suo stato odierno, successivamente sarà presentato nello specifico il gruppo dei servizi al cittadino, la struttura dei team al suo interno e i principali prodotti offerti, questo per arrivare a definire con esattezza le necessità da soddisfare.

- *Capitolo 3*: alla luce delle esigenze emerse, nel precedente capitolo, verranno definiti i requisiti esatti del progetto, analizzando come questi abbiano portato alla volontà di istituire un processo di trasformazione per condurre il gruppo verso l'adozione di un'unica soluzione software. Per proseguire verrà discusso il processo identificato, definendo la politica adottata ed un insieme di passaggi che possono essere proposti per superare questo genere di sfide, ovviamente con i dovuti adattamenti al contesto specifico.
  Una volta definito il processo di trasformazione in maniera rigorosa sarà illustrato l'insieme di analisi e studi effettuati, propri di un approccio a micro-frontend, come descritto nel primo capitolo. Questi, collegati all'identificazione dei servizi proposti ed in linea con le direttive aziendali, hanno portato alla definizione generale del sistema da realizzare.

- *Capitolo 4*: all'interno di questo capitolo verrà definito il processo di automazione legato al ciclo di rilascio del software, in particolare saranno presentate l'organizzazione e le politiche di lavoro in uso, per

concludere con la definizione generale della pipeline CI/CD progettata per i componenti in sviluppo. Tutto ciò solo dopo aver illustrato le principali fasi del processo di integrazione e di distribuzione continua, discutendo i concetti alla base, le tecnologie e i vincoli imposti dall'azienda, aggiungendo varie considerazioni generali in materia, senza l'obiettivo di fornire un quadro completo data la vastità dell'argomento ma approfondendo più nello specifico gli aspetti legati alle architetture a micro-frontend.

- *Capitolo 5*: per concludere verranno presentati due prototipi realizzati, questo per fornire un'idea dei risultati raggiunti e per illustrare quanto è stato possibile dedurre dal loro sviluppo, ossia un insieme di importanti considerazioni, le quali in seguito saranno infatti approfondite per illustrare le soluzioni scaturite dalla loro analisi. Soluzioni pensate per superare i limiti riscontrati ed incrementare la qualità finale del sistema nel suo complesso, oltre che per agevolare le modalità di gestione dello stesso.

Indice

Elenco delle figure

Elenco listati

Capitolo 1

Micro-frontend, stato dell'arte

In questo capitolo verranno presentati i concetti principali legati alle architetture a micro-frontend, sul quale si basa il lavoro di tesi presentato in questo elaborato.
In principio verrà illustrato il contesto dal quale nasce l'esigenza di introdurre questo tipo approccio, successivamente verranno discusse in generale le architetture monolitiche, a microservizi e a micro-frontend, analizzando i principi alla base e le caratteristiche principali. A seguire verranno elencati nello specifico i benefici e gli svantaggi derivanti dall'adozione di un approccio a micro-frontend.
Infine, verranno discusse le modalità con le quali avvicinarsi a questo genere di architetture, in particolare le strategie e le soluzioni attuabili, per proseguire verrà presentato uno dei pattern più comuni che é possibile sfruttare e per concludere alcune considerazioni sulle possibili tecnologie a supporto.

1.1 Introduzione

La nascita del World Wide Web (WWW) viene comunemente fatta risalire al 6 Agosto 1991, giorno in cui l'informatico inglese Tim Berners Lee, presso il Cern di Ginevra, pubblicò il primo sito web [2].
In principio, nel Web era possibile usufruire soltanto di contenuti statici, il

server era responsabile unicamente di rispondere alle richieste che riceveva inviando i file corrispondenti senza effettuare alcun tipo di elaborazione e lato client non poteva essere eseguito codice. I documenti visualizzati erano quindi proposti all'utente nella medesima forma in cui questi erano persistiti lato server.

Da allora il Web si è evoluto enormemente, applicazioni server-side elaborano i contenuti da inviare al browser, queste possono interagire con altri servizi, accedere ad informazioni memorizzate all'interno di un proprio database e generare quindi contenuti dinamici. L'evoluzione però non si è limitata alla componente server, infatti anche lato client abbiamo assistito ad un miglioramento continuo che ha portato dalle prime pagine statiche alla dinamicità delle applicazioni web come le conosciamo oggi.
Questo miglioramento è stato possibile anche grazie allo sviluppo di nuove tecnologie, nel 1995 Netscape Communications[1] ha introdotto JavaScript, un linguaggio di scripting lato client che consente ai programmatori di migliorare l'interfaccia utente e l'interattività grazie ad elementi dinamici. Alcuni anni dopo, nel 2005, Jesse James Garrett ha proposto un approccio allo sviluppo di applicazioni web chiamato AJAX [3], questo consente l'aggiornamento di parti di una pagina web senza doverla ricaricare interamente, ciò ha reso il lato client meno dipendente dal lato server [4].
Dopo il successo di AJAX, il consolidamento sia di HTML5 che degli strumenti utilizzati per migliorare lo sviluppo dell'interfaccia utente, è emerso il concetto di Single Page Application (SPA) [3].

In una SPA, l'intera applicazione viene eseguita all'interno del browser come un'unica pagina web, abbattendo considerevolmente il quantitativo di informazioni scambiate tra frontend e backend. In questo approccio, il livello di presentazione è stato escluso dal server e gestito interamente lato client [5], contemporaneamente lo stesso livello di presentazione di una moderna applicazione web è diventato un componente sempre più importante e cru-

[1] `https://isp.netscape.com/`

ciale.
I team di sviluppo sono quindi costantemente alla ricerca di nuovi modi per sviluppare, distribuire e mantenere le applicazioni in modo efficiente, con l'obiettivo di fornire valore ai proprio utenti in maniera rapida ed efficace.
Nuovi framework frontend vengono introdotti nel mercato frequentemente e gli sviluppatori hanno molte opzioni valide tra cui scegliere per creare applicazioni web potenti e ricche di funzionalità. Tuttavia, la maggior parte di esse finisce per essere composta da frontend monolitici [6], pertanto il lato client dell'applicazione tende a crescere e il suo sviluppo diventa difficile da scalare [7]. Questo fenomeno si riscontra soprattutto nelle aziende più grandi, dove team composti da un numero considerevole di persone devono operare sulla stessa *code base* contemporaneamente.

I micro-frontend sono stati introdotti nel 2016 per consentire la scomposizione del frontend in elementi individuali e semindipendenti, separando la logica applicativa del frontend e creando servizi autonomi capaci di interagire tra loro.
I micro-frontend sono oggi adottati da diverse grandi industrie tra cui DAZN, Ikea, New Relic, SAP, Springer, Starbucks, Zalando e molti altri [6].

1.2 Monolite, microservizi, micro-frontend

L'approccio tradizionale allo sviluppo di un'applicazione consiste nell'adozione di architetture monolitiche. In un'applicazione monolitica, la maggior parte della logica viene rilasciata in un unico artefatto ed eseguita all'interno di un singolo ambiente centralizzato.
L'adozione di questo approccio è però ad oggi in calo, le applicazioni di questo tipo tendono con il passare del tempo, quindi all'aumentare delle dimensioni, a crescere notevolmente in complessità, ciò comporta dei costi sempre maggiori di manutenzione e di evoluzione del prodotto, fino al punto di non consentirne più la gestione entro uno sforzo ragionevole.

Un approccio monolitico porta quindi con sé un certo numero di sfide [8]:

- La *scalabilità* è una delle sfide più grandi che una architettura monolitica porta con sé [9], quando l'applicazione raggiunge dimensioni importanti, il team di sviluppo viene tipicamente suddiviso in più gruppi di lavoro indipendenti, focalizzati su differenti aree funzionali e spesso dislocati sul territorio. Ogni modifica da apportare al sistema necessita quindi di una certa capacità di orchestrazione non indifferente.

- Viste le difficoltà ad apportare nuove modifiche, diventa sempre più complicato *aggiornare o sostituire le tecnologie impiegate*, di conseguenza si promuove la tendenza a progredire utilizzando quelle presenti, pregiudicando le proprie possibilità di sfruttare le nuove opportunità a disposizione sul mercato.

- La presenza di code base di grandi *dimensioni* comporta difficoltà, sia in termini di comprensione del codice sia in termini di performance, in quanto rendono più pesanti e difficili da gestire gli ambienti di sviluppo.

- In un'applicazione monolitica ogni modifica ha la capacità di impattare sull'intero sistema con il rischio di comprometterne le funzionalità, di conseguenza ogni operazione che si desidera effettuare deve essere attentamente analizzata e in seguito collaudata. Effettuare test di non regressione comporta dei costi notevoli rendendo ancora più lento il processo di rilascio, ciò causa una sempre *minor frequenza di aggiornamento*.

Le sfide impegnative, derivanti da un approccio monolitico, hanno portato numerose aziende ad adottare soluzioni che permettessero di scomporre il sistema in componenti di dimensioni ridotte e quindi meno complesse da gestire (Fig. 1.1). Per questo ed altri motivi, si è enormemente diffusa negli ultimi anni la tendenza ad adottare architetture basate su microservizi (Fig. 1.2).

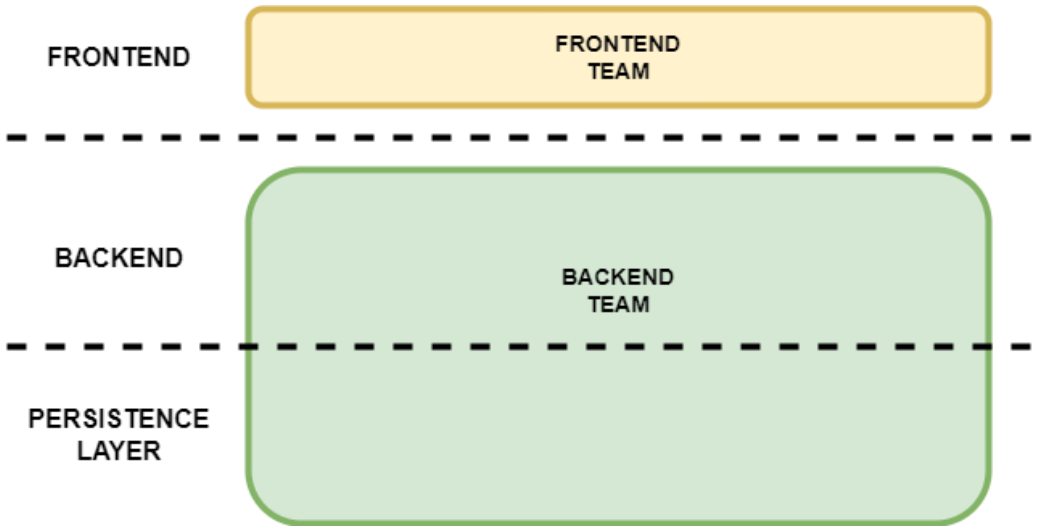

Figura 1.1: Scomposizione di una architettura monolitica con suddivisione in frontend e backend.

In breve, come sostengono Fowler e Lewis [10], una architettura a microservizi consiste in un approccio di sviluppo di singole applicazioni come risultato della composizione di piccoli servizi, ciascuno di essi in esecuzione all'interno del proprio processo e in grado di scambiare informazioni, spesso attraverso API HTTP. Questi servizi sono fortemente disaccoppiati, costruiti in funzione di specifiche logiche di business, inoltre sono rilasciabili ed eseguibili in maniera indipendente.
Le architetture a microservizi si basano sul fornire una maggiore flessibilità e resilienza. Queste due caratteristiche promuovono un cambiamento rapido, il che consente di ricevere feedback continui e quindi di adeguare le strategie di investimento aziendale, al fine di ottenere la massima soddisfazione del cliente ed essere più competitivi sul mercato [8].

Come sostengono Sam Newman e Luca Mezzalira [11][12], alla base di un approccio a microservizi esistono una serie di importanti principi chiave, aderire ad ognuno di essi permette di trarre il massimo vantaggio da questa architettura:

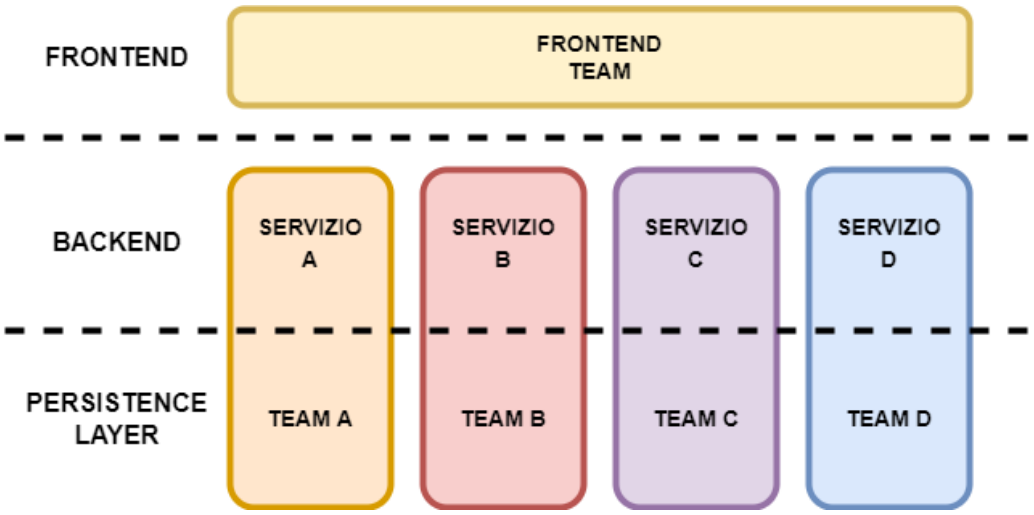

Figura 1.2: Architettura a microservizi.

Adottare una cultura dell'automazione i microservizi aggiungono una complessità non trascurabile al sistema, abbracciare una cultura dell'automazione è un modo chiave per affrontare questo problema, permettendo di muoversi in modo rapido e sicuro. I test automatizzati sono essenziali, poiché garantire che i servizi funzionino ancora è un processo più complesso rispetto a quanto avviene nei sistemi monolitici. Adottare invece tecniche di continuous deployment consente di ricevere feedback rapidi sulla qualità del prodotto ad ogni rilascio.

Nascondere i dettagli implementativi per consentire ai servizi di evolvere indipendentemente dagli altri è vitale nasconderne i dettagli implementativi. I servizi dovrebbero, per esempio, evitare di condividere i propri database, questo al fine di non permettere che modifiche al loro schema si ripercuotano su altri componenti del sistema.

Decentralizzare per massimizzare il livello di autonomia dei microservizi bisogna delegare le decisioni e il controllo ai team che gestiscono i servizi stessi, questo in modo di assicurarsi che le decisioni vengano prese al momento giusto e nel miglior modo possibile in funzione del contesto

legato al problema.

In sistemi basati su architetture monolitiche, la maggior parte delle decisioni chiave vengono prese dalle persone più esperte dell'organizzazione. Queste decisioni, tuttavia, portano spesso a compromessi lungo il ciclo di vita del software. Decentralizzare potrebbe avere un impatto positivo sull'intero sistema, consentendo di evitare l'introduzione di questi compromessi.

Evitare quindi approcci, come enterprise service bus e sistemi di orchestrazione, che portano alla centralizzazione della logica di business.

Distribuibile in modo indipendente nel caso di sistemi monolitici, il rilascio dell'applicativo coinvolge ogni volta l'intero sistema, il rischio di incorrere in problemi in produzione è quindi maggiore e, nel caso sia necessario, effettuare il rollback del sistema richiede tempi più lunghi. Con i microservizi, è possibile effettuarne i rilasci in maniera indipendente, scongiurando la possibilità di creare disservizio dell'intero API layer.

Anche quando non sono necessarie modifiche sostanziali, l'approccio consigliato è quello di garantire la disponibilità delle versioni precedenti, questo per consentire ai consumatori del servizio di adattarsi alle evoluzioni in base ai propri tempi. Grazie a questa gestione è possibile ottimizzare la velocità di rilascio di nuove funzionalità, oltre che aumentare l'autonomia dei singoli team, garantendo che non debbano coordinarsi costantemente in queste fasi.

Adottare questo approccio è oggi ancora più semplice dato che disponiamo di solide tecniche, come le strategie di rilascio canary oppure blue/green, che ci consentono di rilasciare una nuova versione di un microservizio limitando ulteriormente i rischi connessi.

Modellazione attorno alle logiche di business questo principio di modellazione si basa sul concetto di domain-driven design (DDD). L'esperienza ci ha mostrato che interfacce strutturate attorno ad un business-

bounded context sono più stabili di quelle strutturate attorno ai concetti tecnici. Utilizzare questo genere di progettazione permette di raggiungere una maggiore comprensione del sistema e di definire dei precisi confini all'interno del quale i team devono operare.

Isolare il fallimento un sistema a microservizi può essere più resistente ai guasti di un sistema monolitico ma solo se il sistema è stato pensato e progettato per gestire il fallimento. Nei punti in cui si effettuano chiamate di rete è importante evitare di trattare chiamate remote al pari di chiamate locali, poiché ciò nasconderebbe diversi tipi di errori. L'approccio migliore consiste nel presupporre che il fallimento si possa verificare ovunque ed in ogni momento.
Risulta fondamentale: comprendere quando e come utilizzare pattern, come bulkheads e circuit breakers, per limitare le ripercussioni dovute al guasto di un componente. Verificare che i timeout siano impostati in modo appropriato. Comprendere quale sarà l'impatto sull'utente se solo una parte del sistema dovesse adottare comportamenti inattesi. Scoprire quali potrebbero essere le implicazioni di un partizionamento della rete e se sacrificare la disponibilità o la coerenza in una determinata situazione sia la scelta migliore.

Altamente osservabile uno dei motivi per cui preferire una architettura monolitica rispetto ad una a microservizi, consiste nel fatto che è più facile osservare un singolo sistema rispetto ad un sistema suddiviso in più parti. I microservizi offrono molta libertà e flessibilità, ma questo non è gratuito; è importante tenere traccia di tutto tramite log, attraverso di monitoraggio e così via. Mantenere il sistema altamente osservabile è una delle principali sfide dei microservizi.

Luca Mezzalira, all'interno del suo libro Building Micro-Frontends [12], analizza questi stessi principi in rapportato ai micro-frontend, evidenziando come la loro applicazione, sia lato backend che lato frontend, fornisca la soluzione che consente ai team di sviluppo di acquisire la piena padronanza

del dominio aziendale, offrendo un modo semplice di suddividere il lavoro dell'intera organizzazione e di applicare rapidamente miglioramenti continui.

I micro-frontend estendono quindi i concetti dei sistemi a microservizi al mondo frontend [7] trasformando le applicazioni web monolitiche, passando da sistemi single code based a sistemi che combinano insieme più applicazioni frontend di piccole dimensioni.
Ognuna di queste applicazioni può essere eseguita, sviluppata e distribuita in modo indipendente, questo consente ai team di sviluppo di creare servizi isolati e debolmente accoppiati. L'idea alla base dei micro-frontend è quella di gestire un'applicazione web come una combinazione di funzionalità o sottodomini aziendali, in cui ad ogni team può essere assegnato un singolo dominio da gestire (Fig. 1.3) [6].

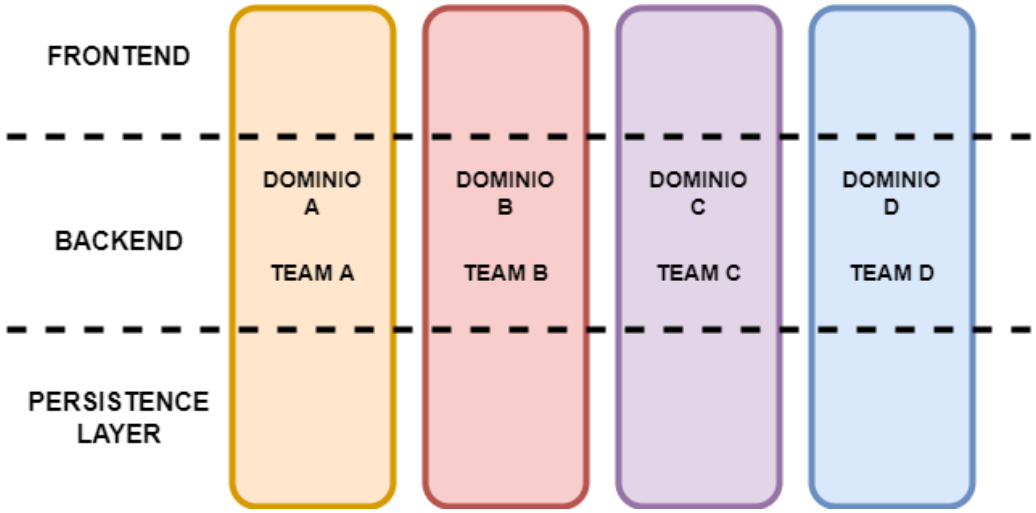

Figura 1.3: Organizzazione end-to-end dei team con architettura a micro-frontend.

In questo contesto è quindi possibile adottare una gestione dei team full-stack, al fine di garantire il pieno controllo dello specifico dominio assegnato, dall'ideazione del livello di presentazione del servizio fino allo sviluppo delle API, della parte di persistenza e quindi dell'intera infrastruttura.

Un pattern che può risultare molto utile in questo contesto è il Backend For frontend pattern (BFF), in cui ogni ad applicazione frontend ne corrisponde una backend, il cui scopo consiste esclusivamente nel soddisfare le esigenze del relativo frontend. Sebbene il pattern BFF in origine sia stato inteso per rappresentare un backend dedicato per ciascun canale frontend (desktop, mobile, ecc.), può essere facilmente esteso al contesto corrente, per indicare un backend per ogni micro-frontend [13].

In conclusione possiamo dire che i micro-frontend condividono i principi chiave dei microservizi, entrambi sono modellati attorno a domini aziendali nascondendo tra loro i dettagli implementativi. Ogni team dovrebbe possedere il proprio microservizio e il relativo frontend, questo consente di decentralizzare le decisioni e di poter effettuare rilasci in maniera indipendente [5].

1.3 Benefici e Svantaggi

All'interno di questa sezione verranno analizzati i maggiori benefici e i principali svantaggi derivanti dall'adozione di architetture basate su micro-frontend. Le informazioni presentate riassumono quanto esposto da S. Pelton, L. Mezzalira e D. Taibi all'interno del loro articolo [6] di revisione su questo tipo di sistemi. Durante la loro indagine hanno provveduto a censire la letteratura grigia e quella accademica sull'argomento, analizzando 173 fonti diverse, di cui 43 in particolare trattavano le motivazioni, i benefici e le problematiche relativamente all'adozione di architetture a micro-frontend.

Di seguito l'elenco dei benefici che le organizzazioni stanno ricevendo dall'utilizzo di un approccio basato su micro-frontend, è interessante notare le analogie con i vantaggi derivanti da un approccio a microservizi:

Supporto per diverse tecnologie con i micro-frontend ogni team di sviluppo può scegliere in autonomia il proprio stack tecnologico, senza

l'obbligo di coordinarsi con gli altri, poiché in questo genere di architetture le soluzioni utilizzate da un determinato team non impattano sugli altri servizi. Questo è un notevole vantaggio ma solleva anche una questione controversa, infatti come dice M. Geers il semplice fatto che sia possibile sceglie una soluzione tecnologica differente non significa che giustifichi farlo [14].

All'interno di un contesto aziendale scegliere delle tecnologie di riferimento e promuoverne l'utilizzo ai propri reparti porta notevoli vantaggi, sia in termini di condivisione delle conoscenze, e quindi di supporto interno, sia in relazione alla facilità di inserimento del personale durante eventuali passaggi da un team ad un altro.

Malgrado in generale non sia consigliabile affidarsi in maniera indiscriminata ad una moltitudine di tecnologie differenti, rimane indubbiamente un notevole vantaggio poter adottare soluzioni alternative in base a particolari esigenze, riscontrabili dai vari team in relazione al loro specifico contesto di applicativo.

Team autonomi interfunzionali come già discusso in precedenza, essendo i micro-frontend autonomi e associabili a un determinato sottodominio applicativo, è possibile dispiegare team di sviluppo interfunzionali e indipendenti. Questo gli conferisce la capacità di prendere decisioni innovative all'interno della propria area di competenza, in quanto il raggio di azione delle loro decisioni è inferiore.

I team interfunzionali hanno il pieno controllo di tutto, dall'ideazione, alla produzione e oltre, questo consente loro di muoversi in maniera rapida ed efficace.

Sviluppo, gestione ed esecuzione indipendenti poiché tutti i moduli che compongono il frontend dell'applicazione sono indipendenti l'uno dall'altro, è possibile svilupparli in maniera autonoma, testarli individualmente e distribuirli più velocemente, effettuando tutto in parallelo si migliorano drasticamente le performance.

Grazie quindi ad una minor necessità di coordinazione con gli altri team, gli sviluppatori possono concentrarsi sul proprio lavoro e fornire rapidamente valore all'azienda.

Migliore testabilità anche la modifica di un piccolo componente, all'interno di un'applicazione monolitica, può avere molteplici effetti collaterali sul resto del sistema. Con i micro-frontend, il test diventa più semplice e rapido, perché lo sviluppatore non deve eseguire l'intera suite di test ogni volta ed inoltre, le modifiche apportate non rischiano di influire sul corretto funzionamento dell'intera applicazione.

Migliore isolamento dei guasti rispetto ad una tradizionale struttura monolitica, in caso di problemi, non viene compromesso l'intero sistema per risolvere eventuali anomalie. Invece, è possibile rilevare il malfunzionamento ed informare l'utente, contemporaneamente si ha la possibilità di intervenire direttamente sul modulo il cui funzionamento risulta compromesso, mentre il resto dell'applicazione continua a funzionare.

Fortemente scalabile grazie alla struttura modulare e all'accoppiamento debole fornito da una architettura a micro-frontend, è possibile suddividere la gestione del frontend su un numero arbitrario di team che possono sviluppare, distribuire, mantenere e utilizzare le proprie soluzioni in modo rapido ed autonomo.

Onboarding più veloce la suddivisione del frontend in più progetti dalla complessità ridotta, richiede una curva di apprendimento minore sul singolo dominio applicativo, questo consente alle nuove risorse di diventare operative in un tempo inferiore.

Caricamento iniziale rapido grazie a questo tipo di architetture è possibile effettuare solo il caricamento dei dati strettamente necessari, demandando il recupero di ulteriori informazioni all'eventuale momento in cui queste risultino effettivamente utili, abbattendo i tempi di caricamento iniziale.

Migliori prestazioni la natura modulare del sistema comporta che l'eventuale ritardo nel caricamento di un micro-frontend, non influisca sulle prestazioni dell'intera applicazione. Inoltre, permette di caricare più velocemente alcune parti di una pagina web, consentendo agli utenti di interagire col sistema prima ancora che tutte le funzionalità vengano completamente elaborate.

Future proof poiché ogni team è ora libero di scegliere la propria tecnologia preferita possiamo definire l'applicazione future proof. I team infatti, non devono più investire su un unico framework ma possono testare le ultime novità presenti sul mercato ed integrarle agevolmente.

Come si può osservare, i vantaggi derivanti da una architettura basata su micro-frontend possono essere molteplici, molti dei quali in comune ad un approccio a microservizi.
In generale il fatto di scomporre un frontend di grandi dimensioni in un insieme di elementi contenuti, leggeri e più maneggevoli, porta numerosi aspetti positivi. I team che partecipano al progetto possono lavorare più velocemente e con una maggiore indipendenza, senza richiedere un investimento di tempo eccessivo per coordinare le operazioni. Inoltre, svincolare il sistema dall'obbligo di doversi legare ad uno stack tecnologico definito, rappresenta un grande vantaggio, specialmente nel lungo periodo.

Come abbiamo visto l'adozione di una architettura basata su micro-frontend porta notevoli benefici, questo approccio porta però con sé anche una serie di sfide con il quale ci si dovrà necessariamente confrontare:

Aumento delle dimensioni del payload utilizzare molteplici framework differenti all'interno di una singola pagina, comporta il download di un quantitativo importante di informazioni, il quale può impattare in maniera negativa sul tempo totale di caricamento iniziale dei contenuti.

Duplicazione del codice bundle JavaScript composti in maniera autonoma possono causare una duplicazione delle dipendenze comuni, aumen-

tando il numero di dati che le applicazioni devono inviare sulla rete agli utenti finali.

Dipendenze condivise una gestione ottimale delle dipendenze condivise aumenta notevolmente le complessità legate alla corretta orchestrazione del sistema.

Moduli diversi possono richiedere versioni diverse delle stesse dipendenze. Qualcuno di essi potrebbe non essere pronto per il passaggio ad una versione superiore, per esempio per il cambio dell'interfaccia, mentre altri potrebbero richiedere il supporto di nuove funzionalità o la correzione di esistenti [15].

Consistenza dell'esperienza utente poiché nuovi framework e librerie di sviluppo in ambito Web vengono rilasciate ad un ritmo sostenuto, la capacità di creare un'esperienza utente (UX) coerente, con interfacce utente avanzate, è più difficile da ottenere. Consentire l'uso di più tecnologie diverse aumenta il rischio di mancanza di consistenza e coesione.

Una soluzione universale, consiste nell'adottare una guida di stile comune. In generale la comunicazione è la chiave per garantire che tutto funzioni senza intoppi, quindi la creazione di una serie di regole e standard comuni può aiutare a ridurre al minimo i conflitti e il rischio di incoerenza.

Monitoraggio il monitoraggio e il debug dei problemi all'intero dell'intero sistema risulta più complesso.

Aumento del livello di complessità dover gestire più sottoprogetti e doversi occupare della loro integrazione aggiunge complessità sia a livello tecnico che organizzativo.

Governance più team che lavorano su un unico prodotto devono essere allineati e avere una visione condivisa, al fine di garantire una unità di intenti a fronte dell'obiettivo finale comune.

Isolamento delle conoscenze questo genere di sistemi porta ad assistere alla presenza di molteplici team interfunzionali che lavorano sullo stesso prodotto, ognuno lavora su una code base diversa e non condivide ciò che sta realmente facendo con gli altri team. In questo contesto è probabile che funzionalità uguali vengano realizzate più volte. Questo rappresenta ovviamente un costo non necessario per l'organizzazione, dispendioso sia economicamente sia in termini di tempo.
Al fine di evitare almeno in parte questo genere di inconvenienti, si consiglia di istituire forme di confronto e condivisione tra i membri dei diversi team.

Differenze tra gli ambienti esistono rischi concreti associati alle differenze tra il contesto in cui viene effettuato lo sviluppo di un componente e l'ambiente finale in cui verrà eseguito, ciò può portare a spiacevoli inconvenienti al momento del rilascio definitivo.
Per limitare questo rischio, è possibile predisporre ambienti di collaudo interni ed adottare tecniche di rilascio come blue/green oppure canary.

Rischio più elevato durante il rilascio di aggiornamenti la maggiore velocità con cui è possibile propagare i cambiamenti riduce la capacità di intercettare eventuali anomalie.

Sfide di accessibilità alcune delle possibili implementazioni di un sistema a micro-frontend, come per esempio tecniche basate sull'uso di iframe, possono comportare problemi di accessibilità.

L'adozione di un approccio basato su micro-frontend comporta quindi un insieme di problematiche non trascurabili. Le organizzazioni che decidono di affidarsi a questa architettura dovranno valutare attentamente le difficoltà, sia dal punto vista tecnico, sia dal punto di vista organizzativo che ne conseguono.
I micro-frontend non possono quindi essere considerati la soluzione definitiva adatta a qualsiasi contesto ma rappresentano un evoluzione architetturale

importante, un utile strumento applicabile allo sviluppo di applicazioni con determinati requisiti lato frontend. In contesti aziendali medio-grandi possono quindi fornire le giuste risposte a varie problematiche riscontrare di frequente.

1.4 Adozione di un approccio a micro-frontend

Al fine di adottare un approccio a micro-frontend esistono varie possibili soluzioni in particolare dal punto di vista tecnologico, prima di addentrarsi in questo aspetto però risulta utile effettuare un insieme di valutazioni.
Il primo quesito che conviene porsi riguarda la granularità degli elementi che si vuole raggiungere attraverso la scomposizione del frontend. Questa operazione può essere eseguita a diversi livelli di dettaglio a seconda del fine che si è interessati raggiungere, inoltre influisce in maniera notevole sui vantaggi e sulle difficoltà che si possono riscontrare dall'adozione dell'architettura.

In relazione a questo contesto, Mezzalira, all'interno del suo libro [12], invita come prima cosa a definire l'approccio che s'intende adottare, dividendolo in una ripartizione del sistema in maniera verticale, caratterizzata dalla presenza di un unico micro-frontend alla volta, oppure orizzontale, che prevede la presenza di più micro-frontend all'interno di una singola pagina.
In una suddivisione orizzontale, la presenza contemporanea di più componenti richiede che i relativi team debbano coordinare i propri sforzi poiché, ciascuno di essi, è responsabile di una parte della vista finale. Malgrado questo approccio richieda una maggiore disciplina ed una capacità di organizzazione superiore, è particolarmente adatto a quelle situazioni in cui è previsto il riuso dei componenti in contesti differenti.
Nel caso di suddivisione verticale invece, ogni team è responsabile di un singolo dominio, in questo contesto risulta quindi molto utile adottare i principi noti dettati da un approccio DDD.

Prima di legarsi a specifiche tecnologie, è inoltre importante svolgere ulteriori considerazioni e scendere nel dettaglio del contesto di utilizzo in base alle specifiche esigenze. Tra gli aspetti principali su cui effettuare i ragionamenti necessari vi sono i seguenti argomenti [16]:

Isolamento valutazione del grado di isolamento dei componenti necessario.

Comunicazione individuazione del livello di interazione necessario tra i vari elementi e quindi del più adatto metodo di scambio di informazioni.

Routing gestione dei percorsi all'interno dell'applicazione e della corretta preparazione delle pagine e dei loro contenuti.

Coerenza della UX gestione di uno stile condiviso al fine di generare un'esperienza utente coerente tra i differenti componenti.

Gestione delle dipendenze introduzione di logiche per ottimizzare il recupero delle dipendenze condivise ed evitare, o quantomeno limitare, caricamenti multipli in pagina.

Composizione valutazione del metodo di integrazione dei componenti in pagina più adatto.

1.4.1 Composizione

Tra i punti elencati al termine della sezione precedente, merita una particolare attenzione la modalità di composizione dei micro-frontend che si desidera adottare.
Con il termine composizione si fa riferimento al processo tramite il quale i componenti vengono recuperati e inclusi dove necessario all'interno della pagina finale. Questa operazione può avvenire anche successivamente rispetto alla generazione iniziale della stessa, inserendo all'interno di essa l'indicazione di quale componente dovrà poi essere recuperato e dove questo andrà collocato.
Il processo può avvenire sia lato client sia lato server. Inoltre, può avvenire

sia in fase di compilazione sia in fase di esecuzione. In quest'ultimo caso è necessario definire una solida strategia di gestione per rendere il sistema scalabile in maniera adeguata, al fine di evitare sovraccarichi e tempi di attesa elevati per gli utenti [12].

Nel caso di una composizione in fase di compilazione, un approccio comunemente adottato consiste nel rilasciare i vari micro-frontend all'interno di un unico pacchetto, i quali dovranno poi essere inclusi come dipendenze all'interno dell'applicazione che dovrà contenerli. Jackson [13] però predilige una composizione in fase di esecuzione piuttosto che questo approccio, in quanto fa giustamente notare che dopo aver organizzato il processo di sviluppo in maniera indipendente non conviene reintrodurre un accoppiamento in fase di rilascio.

In merito ad una gestione basata su composizione lato server in fase di esecuzione una classica tecnica utilizzata, risalente agli anni '90, è il Server Side Include (SSI). Questo approccio non ha subito grosse evoluzioni nel corso degli anni e la sua specifica può quindi definirsi stabile, le sue implementazioni, spesso in combinazione con Nginx[2], sono solide e consentono una valida gestione del carico [14].
Con questo approccio, all'interno di appositi file template, con estensione *shtml*, vengono inserite le istruzioni definite dalla specifica sintassi SSI, come nell'esempio:

```
<!--#include virtual="file.html" -->
```

Questa direttiva viene poi elaborata e sostituita con il contenuto espresso, il risultato finale verrà infine restituito al client e potrà quindi essere mostrato all'interno del browser.
Il metodo mostrato che sfrutta la tecnica SSI è solo una delle possibili soluzioni, un metodo di composizione simile, applicabile tramite l'uso di Content Delivery Network (CDN), consiste nel Edge-side include (ESI) [6].

[2] `https://nginx.org/`

Nel caso in cui la composizione avvenga sempre in fase di esecuzione ma questa volta lato client, l'integrazione dei componenti può essere realizzata tramite varie tecniche. Nel seguito verranno presentate quelle maggiormente note e diffuse [12] [13] [14] [16]:

Integrazione tramite iframe uno degli approcci più semplici per comporre insieme i vari micro-frontend all'interno del browser è quello di inserire ciascuno di essi all'interno di un iframe. Da un punto di vista tecnologico, questa tecnica non è né nuova né entusiasmante, ma possiede il vantaggio di essere facile da implementare e comprendere [14]. Offre inoltre un buon grado di isolamento in termini di stile e variabili globali, in quanto non vi saranno interferenze tra i componenti [13]. La chiave, dal punto di vista dei micro-frontend, è che viene introdotto un accoppiamento minimo tra i team, i quali possono concentrarsi sulla loro missione [14].

L'isolamento facile comporta però anche un insieme di conseguenze che tendono a rendere questa soluzione meno flessibile rispetto ad altre opzioni [13], come l'impossibilità di condividere dipendenze comuni [16] che può portare ad un peggioramento delle prestazioni, ancora rischiano di compromettere l'usabilità dell'applicazione, la flessibilità del layout e la sua reattività, l'accessibilità e la compatibilità con i motori di ricerca [14]. Inoltre, rendono più complicata l'integrazione tra diverse parti dell'applicazione ed il routing [13].

Questo strumento rimane tuttavia indicato per quei contesti dove viene richiesto il massimo livello di isolamento oppure quando è necessario includere un sistema legacy o sviluppato da terzi.

Integrazione tramite JavaScript il concetto alla base di questa gestione è che il necessario per l'esecuzione di ogni micro-frontend viene incluso all'interno di un pacchetto JavaScript distinto, distribuito separatamente e caricato in pagina solo se necessario. L'inclusione in pagina di questo pacchetto consente, all'applicazione che svolge il ruolo di contenitore, di richiedere la visualizzazione dei contenuti invocando

un'apposita funzione JavaScript, definita come punto di accesso per il componente.

Adottando questa soluzione è possibile caricare librerie e stili comuni senza duplicazioni, riducendo la dimensione finale dell'applicazione [16]. Si tratta di un approccio robusto, facile da implementare e che garantisce un buon livello di flessibilità, permettendo di creare integrazioni tra i micro-frontend in vari modi [13].

A differenza della gestione basata su iframe, i contenuti vengono inseriti in pagina senza il supporto di specifiche tecniche di isolamento, questo può portare a comportamenti inattesi dati da possibili conflitti, in primis a livello di stile. Per evitare questo genere di spiacevoli inconvenienti è consigliabile adottare apposite convenzioni durante l'attribuzione dei nomi ad ogni singolo elemento [14].

Integrazione tramite Web Component una variante della soluzione proposta in precedenza consiste nel definire, per ogni micro-frontend, un Web Component istanziabile, all'interno dell'applicativo che svolge il ruolo di contenitore, come alternativa alla definizione di una funzione globale da richiamare. L'obiettivo è quello di introdurre un migliore incapsulamento e consentire l'interoperabilità tra diverse librerie o framework all'interno dell'applicativo.

Questo approccio è realizzabile grazie allo standard HTML emesso dal World Wide Web Consortium (W3C) [17], che consente di creare elementi HTML personalizzati, definire il loro comportamento e caricarli in modo dinamico [16]. I Web Component sono costituiti da più tecnologie separate che vengono utilizzate assieme, tra queste vi sono Custom Element[3] e Shadow DOM[4].

Un Custom Element può incapsulare la relativa logica applicativa e fornire l'interfaccia utente associata. Mediante l'uso di Shadow DOM invece, è possibile separare una parte del DOM dal resto della pagi-

[3] `https://html.spec.whatwg.org/multipage/custom-elements.html`

[4] `https://dom.spec.whatwg.org/#shadow-trees`

na. Questo permette di isolare CSS e JavaScript quasi come attraverso l'uso di un iframe, evitando conflitti di stile ed aumentando quindi la robustezza dei micro-frontend sviluppati ma senza dover adottare particolari convenzioni nell'assegnazione dei nomi.
Il vantaggio più significativo dell'utilizzo di Web Component come tecnica di integrazione è che sono uno standard Web ampiamente adottato, ciò comporta una loro lenta evoluzione ma soprattutto previene dall'introduzione di cambiamenti radicali, o comunque tali, da renderli non compatibili con le versioni precedenti.
Custom Element e Shadow DOM forniscono entrambi maggiori possibilità in termini di isolamento non raggiungibili mediante l'uso di altre tecniche, anche se utilizzare entrambi gli standard non è obbligatorio.
I metodi del ciclo di vita definiti dalla specifica legata ai Custom Element consentono di definire un'interfaccia omogenea per interagire con i componenti. Senza questo standard, i team dovrebbero concordare una procedura per l'inizializzazione, la deallocazione e l'aggiornamento dei componenti.
Uno dei punti critici più importanti legati all'uso di Web Component è che, per funzionare, richiedono l'esecuzione di codice JavaScript lato client, pertanto non possono essere pre-elaborati lato server, questo comporta una maggiore lentezza durante il caricamento iniziale della pagina.
Il supporto del browser per i Web Component è notevolmente migliorato negli ultimi anni, integrare i Custom Element sui browser meno recenti non è complicato, eseguire il *polyfill* per poter applicare lo standard definito Shadow DOM è più complesso [14].
In conclusione, i Custom Element forniscono una valida soluzione per incapsulare i propri componenti JavaScript e renderli accessibili in modo standard. Shadow DOM introduce un meccanismo di isolamento aggiuntivo, riduce il rischio di interferenze e può anche aiutare ad integrare in sicurezza un micro-frontend in un sistema legacy, il tutto

senza generare conflitti di stile. È possibile creare applicazioni altamente interattive con una composizione lato client in fase di esecuzione ma siccome necessitano dell'esecuzione di codice JavaScript per funzionare, una gestione che prevedere un'integrazione lato server fornirà una soluzione mediamente più veloce durante il caricamento iniziale della pagina [14].

Gli esempi di composizione trattati, mostrano come l'introduzione di una architettura a micro-frontend non richieda necessariamente di formarsi su nuove tecniche e la loro adozione non debba per forza essere eccessivamente complessa. Il motivo per cui è possibile definirli tali, risiede nella suddivisione effettuata a livello di dominio e nell'autonomia conferita ai singoli team in fase di sviluppo, oltre che alla definizione di pipeline di rilascio indipendenti. Adottando le giuste tecniche di progettazione è possibile ottenere i vantaggi dichiarati indipendentemente dallo stack tecnologico utilizzato [13].

1.4.2 Comunicazione

Uno degli interrogativi più comuni che ci si pone quando si approccia una architettura a micro-frontend, consiste nel come poter mettere in comunicazione tra loro le varie parti del frontend e favorire quindi un corretto scambio di informazioni tra i componenti stessi ed eventualmente anche l'applicazione container.
In generale vi sono più modi per rispondere a questo problema, nel seguito verranno prese in esame alcune strategie adottabili ed effettuate alcune considerazioni su di esse, al fine di comprendere quelle che saranno le possibili soluzioni applicabili all'esigenza in funzione del contesto effettivo.

Una delle strategie principali di comunicazione tra micro-frontend è quella tramite backend. A fronte dell'interazione con l'utente, ogni micro-frontend può infatti persistere le richieste ricevute effettuando delle chiamate a backend, successivamente ogni elemento interessato a queste informazioni sarà in grado di recuperarle facilmente invocando gli stessi servizi di persistenza.

Questo approccio ha il vantaggio di essere semplice e di non richiedere la definizione di un'interfaccia di comunicazione condivisa lato client. Coinvolgere il lato server comporta però un aumento di latenza e quindi una minor reattività, inoltre non è una soluzione adatta ad ogni contesto e a tutte le tipologie di informazioni.

Tra le strategie di comunicazione lato client è consigliato evitare una forma di comunicazione diretta tra componenti, in quanto può condurre ad un accoppiamento elevato tra di essi [14], preferibile invece una forma di comunicazione basata su canali condivisi.
In questo caso viene predisposto un meccanismo che consenta a componenti disaccoppiati di comunicare tra loro tramite eventi, inviati sfruttando un bus condiviso. Ogni micro-frontend interessato può quindi decidere se reagire o meno a determinate tipologie di eventi, in base a come ritiene più opportuno.
Un esempio di questa gestione è rappresentato da una comunicazione basata su Broadcast Channel API[5]. Infatti, creando un oggetto `BroadcastChannel`, è possibile ricevere tutti i messaggi che vengono pubblicati su di esso, consentendo una comunicazione bidirezionale tra tutti coloro che ne sono interessati.
Una soluzione simile a questa comporta invece l'uso di Custom Event[6] per generare e reagire ad eventi rappresentati dall'interfaccia `CustomEvent`. In questo modo oltre agli eventi più classici, come quelli generati da mouse e tastiera, è possibile definirne di personalizzati per far fronte alle proprie specifiche esigenze.

Come mezzo per condividere informazioni si fa spesso affidamento a strutture comuni, alcune informazioni vengono trasmesse tramite header HTTP oppure tramite cookie, altre volte si sfrutta l'URL della pagina, inserendo informazioni in query-string o all'interno del percorso della risorsa.
In altri casi si sfrutta il web storage oppure si fa affidamento a librerie JavaScript pensate per condividere informazioni all'interno dell'applicazione,

[5] `https://developer.mozilla.org/en-US/docs/Web/API/Broadcast_Channel_API`

[6] `https://developer.mozilla.org/en-US/docs/Web/Events/Creating_and_triggering_events`

come per esempio Redux[7]. È importante sottolineare che l'uso di soluzioni di questo tipo deve essere fatto con criterio, evitando di creare uno stato condiviso.
Allo stesso modo in cui si sconsiglia di condividere database tra microservizi diversi, si sconsiglia di creare uno stato comune tra micro-frontend. In quanto condividendo strutture dati e modelli di dominio, si instaura un forte accoppiamento tra i componenti, il che rende estremamente difficile apportare modifiche in futuro [13].

Infine un metodo efficace consiste nell'assegnare, all'elemento padre, che svolge il ruolo di contenitore all'interno dell'applicativo, il ruolo di orchestratore della comunicazione, instaurando una forma di comunicazione verticale all'interno del sistema.
Questo approccio è collegato alla strategia di composizione impiegata, per esempio nel caso di una soluzione basata su Web Container è possibile sfruttare il metodo `attributeChangedCallback`, richiamato quando uno degli attributi viene modificato.
In generale, è consigliato definire un'interfaccia standard che non richieda modifiche strutturali ogniqualvolta si presenti la necessità di definire un nuovo tipo di messaggio da scambiare tra due micro-frontend.

In conclusione è bene far notare che la presenza di un flusso di informazioni troppo consistente può essere indice di una definizione del dominio non accurata, infatti ogni micro-frontend dovrebbe essere in grado di operare in maniera autonoma, rimanendo all'oscuro della presenza di altri componenti all'interno della pagina.
Indipendentemente da quali e da quante strategie diverse di comunicazione si decida di adottare all'interno del sistema, è fondamentale valutare con attenzione le scelte fatte, al fine di non instaurare un accoppiamento forte tra componenti e quindi evitare di introdurre forme di dipendenze che possono portare ad una riduzione dei vantaggi attesi dall'adozione dell'architettura.

[7] `https://redux.js.org/`

1.4.3 Backends For Frontends pattern

In materia di comunicazione tra micro-frontend e backend vi sono vari pattern noti che possono essere adottati, due di questi molto famosi e diffusi si basano sull'utilizzo di API gateway e Service dictionary
Un altro pattern che può essere utilizzato in questo contesto, grazie alla separazione netta che introduce tra i client e le server API, è noto con il nome di Backends For Frontends (BFF) [12].

Il pattern BFF è stato introdotto in origine in seguito alla diffusione dei dispositivi mobile, quando servizi server-side, pensati per essere consumati da interfaccia web desktop, hanno cominciato a dover far fronte alle esigenze di questo differente tipo di dispositivi. In molti casi realizzare questa integrazione ha causato molti problemi, per via di un accoppiamento stretto tra l'interfaccia utente web desktop e i servizi esistenti.
La situazione è ancora più complicata da gestire in tutte quelle realtà di dimensione medio elevata dove la gestione di frontend e backend è suddivisa tra più team diversi. In questi contesti il team di backend ha dovuto cominciare ad interfacciarsi e a far fronte alle esigenze di un secondo team di frontend, stando attento a bilanciare e a dare la giusta priorità alle necessità di entrambi i propri interlocutori [18].

In questo genere di situazioni è possibile adottare il pattern BFF, in cui viene spezzata questa dipendenza diretta, in quanto ad ogni applicazione frontend viene fatto corrispondere un backend, con lo scopo preciso di soddisfare le sue esigenze specifiche.
Il componente BFF è quindi strettamente collegato ad una singola esperienza utente e gestito dallo stesso team che si occupa della relativa interfaccia. Questo rende meno complicato fornire al frontend le API adatte e semplifica anche il processo di rilascio dei componenti client e server [18].
Un ulteriore vantaggio conseguibile con questo approccio, consiste nel fatto di ridurre i messaggi scambiati tra frontend e backend, ciò è possibile ag-

gregando più risposte di diversi servizi in un'unica struttura dati, costruita in maniera tale da essere facilmente consumata dal frontend in base alle sue caratteristiche specifiche, utile per ridurre il numero di messaggi scambiati tra client e server [12].
In generale vi sono più modi di adottare questo pattern, il BFF potrebbe essere autonomo, caratterizzato da una proprio logica di business e dotato di un proprio database, oppure potrebbe limitarsi a svolgere il ruolo di aggregatore di servizi e, nel caso, potrebbe aver senso che questo venga gestito dallo stesso team che possiede il frontend [13].

In particolare all'interno di un contesto composto da una architettura a microservizi e a micro-frontend è possibile fare alcune osservazioni ulteriori relative al pattern. Infatti, è interessante notare come la complessità di una architettura a microservizi, anche di grandi dimensioni, possa essere nascosta dietro ad un unico punto di accesso, ossia il BFF, sollevando il frontend dal dover necessariamente comprendere questa complessità [12].
Inoltre, sebbene il pattern sia stato originariamente concepito con l'idea di affiancare un componente di backend ad ogni client, questo per ogni tipologia di dispositivo (desktop web, mobile, ecc.), esso può essere facilmente esteso per indicare un backend per ogni micro-frontend [13].
Seguendo questo concetto, il dominio principale può essere scomposto in più sottodomini identificati da un unico punto di ingresso, raggruppando i microservizi per ognuno di essi invece di prendere in considerazione il tipo di dispositivo che dovrebbe consumare le API. All'interno di ogni singolo dominio è possibile gestire con una maggiore coerenza la risposta di ogni frontend, consentendo anche all'applicazione di gestire gli errori in maniera più ordinata, garantendo una maggior tolleranza ai guasti rispetto ad un sistema composto da un unico livello responsabile di servire tutte le API [12].

In figura 1.4, viene illustrato questo concetto, dove per due domini differenti, che possono condividere alcuni servizi, vengono esposte a micro-

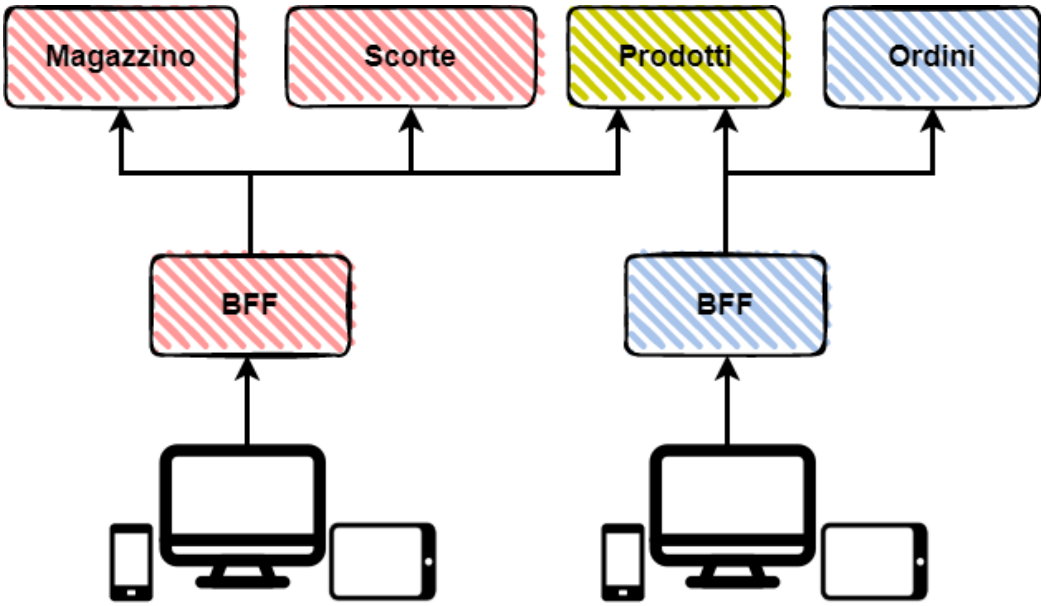

Figura 1.4: Applicazione del pattern Backends For Frontends per dominio.

frontend diversi, API composte dall'eventuale aggregazione di servizi di dominio.
Un importante vantaggio di un sistema costruito in questo modo, consiste nel fatto che permette ai componenti BFF di scalare dinamicamente, in maniera agile, in base allo specifico traffico ricevuto [12].

1.4.4 Framework a supporto

L'adozione di architetture basate su micro-frontend si sta diffondendo notevolmente nell'industria del software, per questo motivo si sta assistendo in quest'ultimo periodo alla nascita di numerosi framework e librerie a supporto di questo genere di sistemi.
Vista la continua evoluzione di questi strumenti e la frequente apparizione di nuove alternative, si è ritenuto opportuno non approfondire nessuna di esse all'interno di questo capitolo ma piuttosto fornire una breve panoramica delle tecnologie attualmente disponibili.

Esistono quindi varie soluzioni nate con il preciso scopo di assistere gli sviluppatori nell'adozione di una architettura a micro-frontend, durante l'implementazione di applicazioni web su larga scala, un esempio di questi sono Mosaic[8] ed OpenComponents[9] oltre che Bit[10] ed Ara Framework[11].
Tra i principali strumenti a supporto, molti di questi si basano sullo standard già introdotto in sessione 1.4.1, ossia i Web Components, proprio per via del fatto che si tratta di uno standard Web esistono librerie e tutorial per tutti i framework JavaScript più diffusi.
Angular, per esempio, viene fornito con una funzionalità chiamata Angular Elements. Questa funzione permette di generare automaticamente il codice necessario per integrare il componente con i metodi del ciclo di vita dei Custom Elements, inoltre supporta l'uso di Shadow DOM. Invece, Vue.js fornisce una soluzione simile tramite il pacchetto ufficiale *@vue/web-component-wrapper*.
Uno strumento, sempre di questo tipo, molto diffuso è Stencil[12] pensato per la creazione di Design System riutilizzabili e scalabili. Google invece, a questo scopo ha prima proposto il progetto Polymer[13], che è stato poi sostituito da Lit[14].
Le soluzioni a disposizione non terminano qui, infatti esistono altri strumenti come Luigi[15], Webpack 5 Module Federation Plugin[16] e Single-Spa[17].

Come si può facilmente intuire da questa brevissima panoramica, gli strumenti a supporto sono numerosi e ne esistono molti altri che non sono stati citati. Infatti, piuttosto che soffermarci su una lunga ed attenta valutazione

[8] `https://www.mosaic9.org/`
[9] `https://opencomponents.github.io/`
[10] `https://bit.dev/`
[11] `https://ara-framework.github.io/website/docs/nova-architecture`
[12] `https://stenciljs.com/`
[13] `https://polymer-library.polymer-project.org/`
[14] `https://lit.dev/`
[15] `https://luigi-project.io/`
[16] `https://webpack.js.org/concepts/module-federation/`
[17] `https://single-spa.js.org/`

di questo settore in evoluzione, nelle precedenti sessioni, ci si è focalizzati su un insieme di tecniche ed argomenti utili a prescindere dai possibili framework impiegabili, concentrandoci sugli standard Web esistenti e sulle funzionalità native dei browser.
Questo per il semplice fatto che, anche se in seguito si dovesse decidere di adottare strumenti di supporto di questo tipo, comprendere i concetti fondamentali alla base di questa architettura è imprescindibile per la creazione di sistemi di successo.

Capitolo 2

Il caso di studio

In questo capitolo verrà riportato il contesto del caso di studio che avrà luogo all'interno dell'azienda Maggioli S.p.A, sarà quindi presentata l'evoluzione che ha subito l'azienda nel tempo, verranno descritte le sue caratteristiche principali e il suo stato attuale. Saranno presentati i prodotti e i servizi offerti, in particolare quelli più significativi per il caso di studio in corso. Infine, verrà analizzato lo stato corrente dei servizi al cittadino offerti per arrivare poi a definire le esigenze al quale è necessario fare fronte.
Questo al fine di individuare la soluzione architetturale migliore per le evoluzioni future, in grado di soddisfare le esigenze di mercato attuali e quelle che si andranno a delineare.

2.1 Gruppo Maggioli

Il Gruppo Maggioli opera principalmente nel settore della Pubblica Amministrazione ma offre anche servizi dedicati ad aziende e liberi professionisti. Il mercato di riferimento per l'azienda è quello nazionale ma di recente il gruppo ha affermato la sua presenza in Spagna e ha dimostrato interesse anche per il mercato del Sud America.

2.1.1 Storia

Le radici del gruppo si fondano sulle attività imprenditoriali avviate dai membri della famiglia Maggioli, con la produzione di prodotti artigianali a partire dal 1905. Negli anni seguenti l'azienda diviene fornitrice del Ministero dell'Educazione Nazionale e successivamente entra nel settore della tipografia. Nel 1972 viene fondata la casa editrice che nel 1978 prenderà il nome di Maggioli Editore, specializzata in prodotti editoriali e servizi professionali per liberi professionisti e Pubblica Amministrazione. Nel 1982 nasce Maggioli Formazione che propone corsi di formazione indirizzati ad amministratori e funzionari degli enti locali [19].
A partire dal 1988 entra a far parte del gruppo Maggioli Informatica, con essa prende il via il processo di sviluppo tecnologico che investe l'intera società ed arricchisce la sua offerta con software gestionali, servizi e progetti per l'informatizzazione delle imprese [19]. Negli anni successivi l'azienda prosegue la propria espansione tramite una serie di acquisizioni e partecipazioni societarie strategiche. A partire dal 2016 allarga ufficialmente i propri orizzonti espandendosi oltre i confini nazionali, in America Latina con Maggioli Latam, in Spagna con Infaplic Sa e Atm Sl, che verranno successivamente fuse con Galileo Sa (Tenerife, Isole Canarie) per formare ATM Grupo Maggioli [20], ed in seguito in Grecia con l'apertura di una nuova sede ad Atene.

Attualmente il Gruppo Maggioli costituisce un'importante realtà nazionale, il suo sviluppo è proseguito costantemente nel tempo andando ad accrescere l'azienda sia in termini di fatturato che in termini di offerta proposta ai proprio clienti.
La crescita del gruppo è continuata anche nel 2019, tramite l'inizio della collaborazione, culminata poi con l'acquisizione, dell'azienda GLOBO, dotata di consolidata esperienza nel campo dei Sistemi Informativi Geografici (GIS). Successivamente, altre operazioni di rilievo effettuate, sono state tre importanti acquisizioni che riguardano, Deepcyber, Injenia e Sinapsys. Queste operazioni hanno permesso al gruppo di arricchire le proprie compe-

tenze in materia di cyber security, intelligenza artificiale, sistemi antifrode, cloud transformation, data management e machine learning, con un focus su soluzioni tecnologiche in ambito sanitario [19].

2.1.2 Profilo della compagnia

Attualmente il Gruppo Maggioli conta 140.000 clienti e 2.400 collaboratori di cui 250 sviluppatori, è presente sul territorio nazionale ed internazionale, possiede 10 centri di ricerca e sviluppo e 70 sedi e filiali [19], la cui posizione geografica viene illustrata nell'immagine 1.3.

Figura 2.1: Mappa delle sedi del Gruppo Maggioli [21].

Il bilancio consolidato del 2020 del Gruppo Maggioli raggiunge i 186 milioni di euro di ricavi, con la previsione per il 2021 di superare i 210 milioni, inoltre, in aumento rispetto alla classifica dell'anno precedente, Maggioli S.p.A. si posiziona al settimo posto della classifica delle Top 100 Aziende di Software e Servizi IT 2020 di Data Manager per quanto riguarda il settore PA [22].
La sede di riferimento dell'area informatica di Maggioli si trova a Santarcangelo di Romagna (RN), di fronte alla sede legale situata all'indirizzo via del Carpino 8.

2.1.3 Prodotti e Servizi

Il Gruppo Maggioli, oltre ai servizi storici di editoria, modulistica e formazione offerti, possiede un ampio catalogo di prodotti e servizi di categoria IT. Il prodotto di punta di Maggioli Informatica è Sicr@web, una suite modulare pensata per soddisfare le esigenze delle amministrazioni locali.

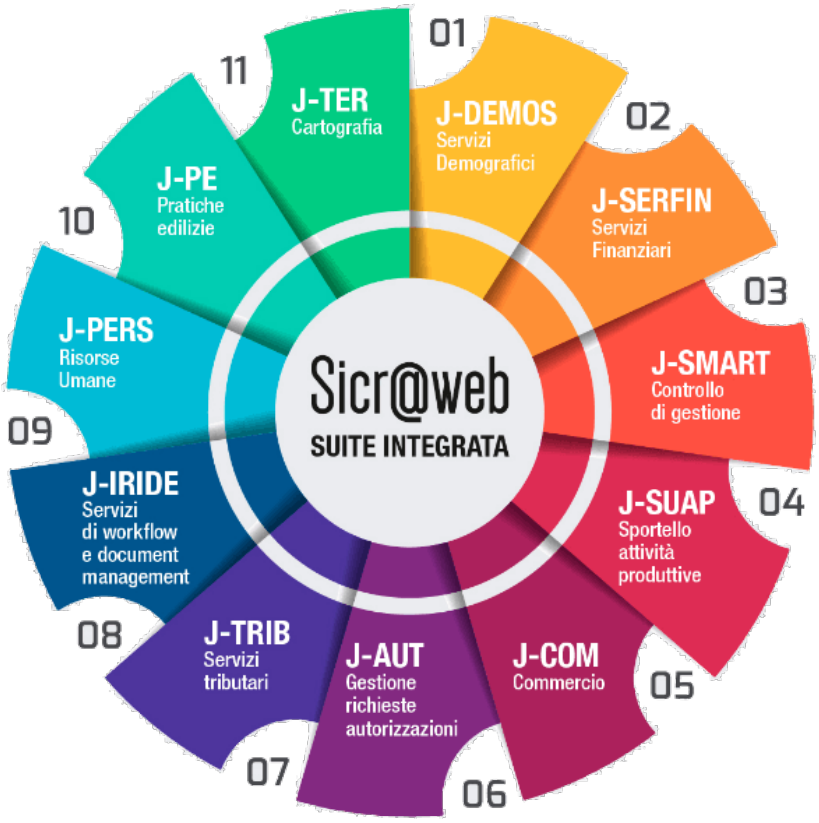

Figura 2.2: Mappa dei moduli gestionali di Sicr@web [19].

Caratteristica principale del software consiste nella capacità di integrare le informazioni non solo tra i vari uffici dell'ente locale ma anche con altri enti, con privati e con i servizi di gestione documentale. Un elenco dei moduli del gestionale sono riportati in figura 2.2, tra i principali troviamo, J-Demos per la gestione dei servizi demografici, J-Trib per la gestione dei tributi, J-PE

per la gestione delle pratiche edilizie e J-Iride per la gestione e semplificazione dei flussi documentali e dei procedimenti amministrativi.

Un ulteriore importante funzionalità di Sicr@web è la possibilità di integrarsi con altri prodotti di Maggioli a supporto delle pubbliche amministrazioni, tra questi possiamo trovare Concilia, pensata per le attività dei comandi di polizia locale, che permette la gestione delle violazioni al Codice della Strada ed inoltre Icaro, pensato per la gestione dei servizi socio-assistenziali e socio-sanitari. Altro prodotto rilevante, integrabile con Sicr@web, è J-City Gov, una piattaforma web altamente personalizzabile, pensata per l'interazione dei cittadini e delle imprese con l'ente locale, J-City Gov appartiene quindi alla categoria di prodotti offerti denominati come servizi al cittadino.

2.2 Servizi al cittadino

Con il processo di digitalizzazione delle pubbliche amministrazioni ha preso sempre più forma l'esigenza per gli enti locali di comunicare in maniera rapida ed efficace con i propri cittadini. Al fine di semplificare il rapporto tra i cittadini e l'amministrazione locale, ridurre i tempi di attesa ed alleggerire il carico di lavoro degli uffici, Maggioli offre un insieme di prodotti, tra cui i principali sono J-City Gov, Municipium App e lo Sportello telematico polifunzionale.

L'evoluzione e la manutenzione di questi prodotti all'interno dell'azienda è in carico ad un unico gruppo che si suddivide in tre team, attualmente a ciascuno di questi è affidata la gestione di uno degli applicativi precedenti. I tre team sono dislocati sul territorio nelle sedi aziendali di Santarcangelo di Romagna (RN), Treviolo (BG), Forlì e Cesena (FC).
La distribuzione del gruppo su più sedi deriva dall'evoluzione che ha avuto nel tempo ed in particolare all'espansione dell'intera azienda nel corso degli anni. Infatti, come già annunciato, Maggioli ha compiuto varie acquisizioni strategiche che le hanno permesso di consolidarsi nel tempo come uno dei

principali player del settore Information & Communication Technology a supporto della Pubblica Amministrazione.

Il gruppo dei servizi al cittadino all'interno di Maggioli, prese forma quando venne deciso di avviare la creazione di un unico prodotto per rispondere alle esigenze dei cittadini. L'obiettivo era quello di sostituire i prodotti, come Uliss-e e Polis, che costituivano l'offerta proposta in quel momento storico per questo genere di servizi, prodotti che provenivano dall'acquisizione di altre aziende. L'idea era quella di promuovere un'unica soluzione, integrata con i gestionali aziendali di back-office, in grado di rispondere a tutte le esigenze che i precedenti software erano in grado di soddisfare, per poi ampliare la propria offerta arricchendola negli anni a venire in base alle esigenze di cittadini ed amministrazioni locali.

Questo prodotto prese il nome di J-City Gov, fu rilasciato nel 2013 e i membri del gruppo di lavoro operavano esclusivamente nella sede di Santarcangelo di Romagna. Nella sua prima versione il software conteneva esclusivamente le funzionalità di gestione delle imposte di soggiorno ed il servizio di Amministrazione Trasparente. Quest'ultimo servizio nacque per rispondere al principio della trasparenza, inteso come accessibilità totale alle informazioni che riguardano l'organizzazione e l'attività delle pubbliche amministrazioni.
Mucipium App entrò a far parte dei prodotti della suite successivamente. Realizzato nel 2015, è stato pensato come un servizio per dispositivi mobili per favorire la comunicazione tra amministrazione locale e cittadini.
Infine, l'offerta di servizi al cittadino di Maggioli si è allargata ulteriormente nel 2020 con lo Sportello telematico polifunzionale, questo a seguito dell'incorporazione di GLOBO all'interno del gruppo Maggioli. GLOBO rappresentava uno dei principali competitor dell'azienda in materia di servizi al cittadino. Infatti, J-City Gov e Sportello telematico polifunzionale condividono alcune funzionalità, in particolare il servizio di presentazione pratiche, il quale consente agli utenti di compilare online, in modo semplice e guidato

i moduli digitali, firmarli dove necessario, integrarli con gli allegati richiesti e trasmetterli agli uffici competenti.

2.2.1 J-City Gov

Il progetto J-City Gov, lato produzione, è attualmente gestito da un team composto da sei sviluppatori, ripartiti sulle sedi di Santarcangelo di Romagna e Forlì.
L'Applicativo, come anticipato, fu rilasciato nel 2013 e nel corso del tempo è stato arricchito con varie funzionalità per rispondere alle principali necessità delle amministrazioni locali e dei loro cittadini. In particolare nel 2016, a seguito degli importanti investimenti fatti e degli sforzi profusi anche durante l'anno precedente, ha registrato la sua maggior crescita di mercato aumentando notevolmente la sua base di utenti. Questo è dovuto anche grazie al rilascio di nuove funzionalità come la gestione dei pagamenti, il modulo di polizia locale, l'arricchimento del Sistema di Gestione e Presentazione Istanze (SGPI) con i pacchetti SUE (Sportello Unico per l'Edilizia) e SUAP (Sportello Unico per le Attività Produttive) e la funzionalità di autenticazione tramite SPID.

Uno dei principali vantaggi di J-City Gov consiste nella profonda integrazione che possiede con gli altri prodotti aziendali, in figura 2.3 viene riportata la struttura dell'applicativo, indicando anche lo strato di connettori che permettono appunto di operare con gli altri applicativi, nell'esempio in questione con Sicr@web.
J-City Gov è quindi strutturato in modo tale che vi sia uno strato orizzontale condiviso, il quale definisce tutte le funzionalità generali indispensabili al funzionamento del sistema, tra cui appunto i connettori. Sopra a questo strato sono situati un insieme di moduli applicativi pensati per offrire una determinata categoria di servizi, tra i principali attualmente a disposizione troviamo:

- Amministrazione trasparente.

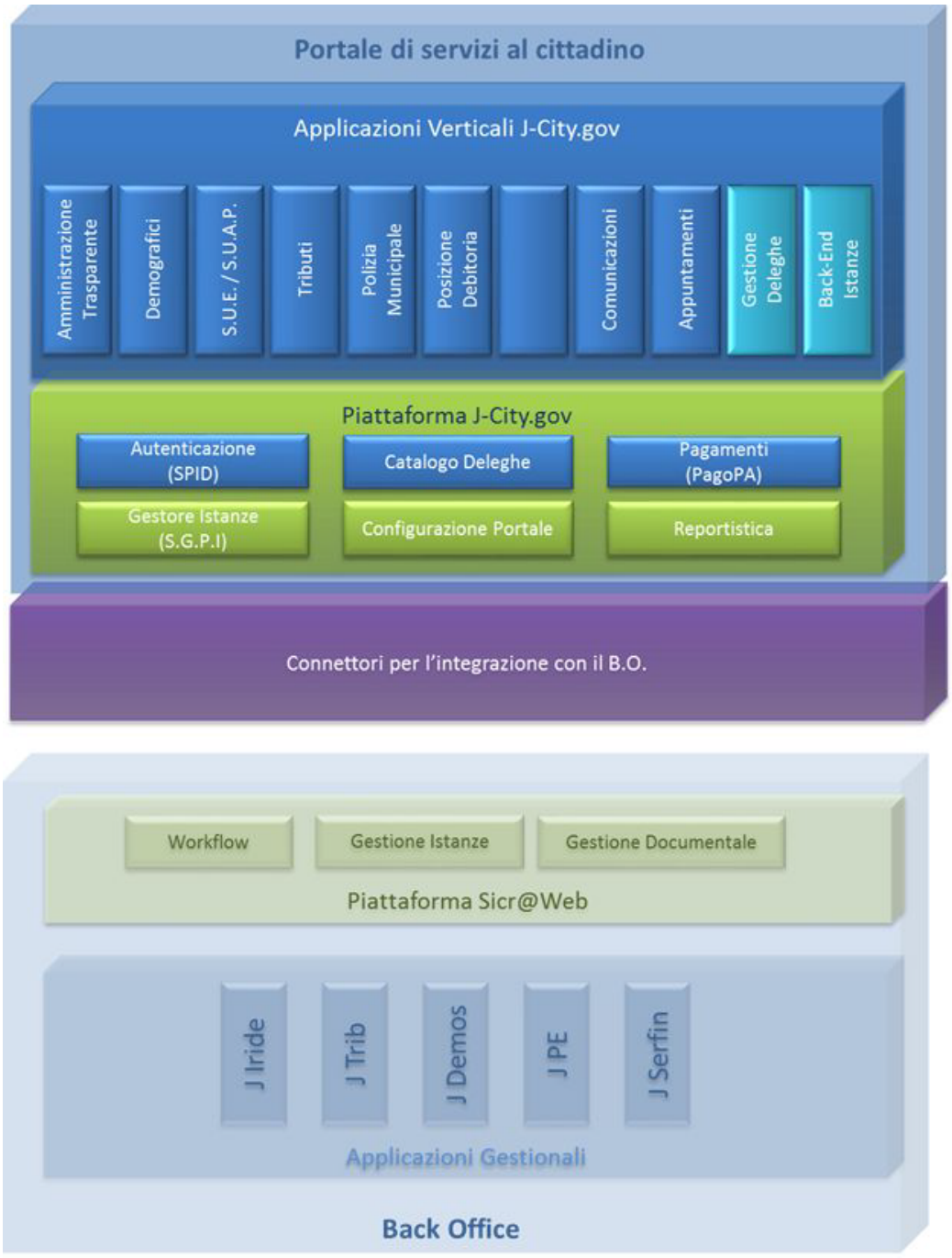

Figura 2.3: Rappresentazione della struttura di J-City Gov e del suo contesto (fonte: documento aziendale interno).

- Albo pretorio.
- Servizi demografici.
- Gestione tributi.
- Polizia locale.
- Sistema di Gestione e Presentazione Istanze (SGPI).
- Servizio prenotazioni.
- Servizio pagamenti.

J-City Gov è un progetto sviluppato sulla piattaforma Liferay Portal, in particolare sulla sua versione 6.2. Liferay è un Enterprise Portal Open Source scritto in Java, distribuito con licenza GNU LGPL e opzionalmente con licenza commerciale.
Nel 2021 è stato riconosciuto da Gartner come leader, per l'undicesimo anno consecutivo, nel Magic Quadrant per le Digital Experience Platforms (DXP). Nel report di Gartner dedicato ai technology decision maker, sono stati analizzati 16 fornitori di Digital Experience Platforms, i quali sono stati valutati per la loro abilità di esecuzione e completezza di visione. Liferay è stato premiato per la sua capacità di integrazione, il basso costo di gestione e la sua capacità di adattamento ai casi complessi B2B e B2E [23].

Liferay Portal utilizza i principi di progettazione dell'architettura Service Oriented (SOA). Inoltre è stato pensato per distribuire componenti basati sul linguaggio Java Enterprise Edition e sulla tecnologia Portal Server Liferay, conforme alle specifiche definite dallo standard JSR-168 e JSR-286 [24]. Una serie di portlet vengono fornite immediatamente insieme al portale, tra queste troviamo incluse quelle adibite alle gestione di Documenti e Media, calendario, blog e wiki, oltre a queste è possibile ovviamente aggiungere le proprie personalizzazioni.

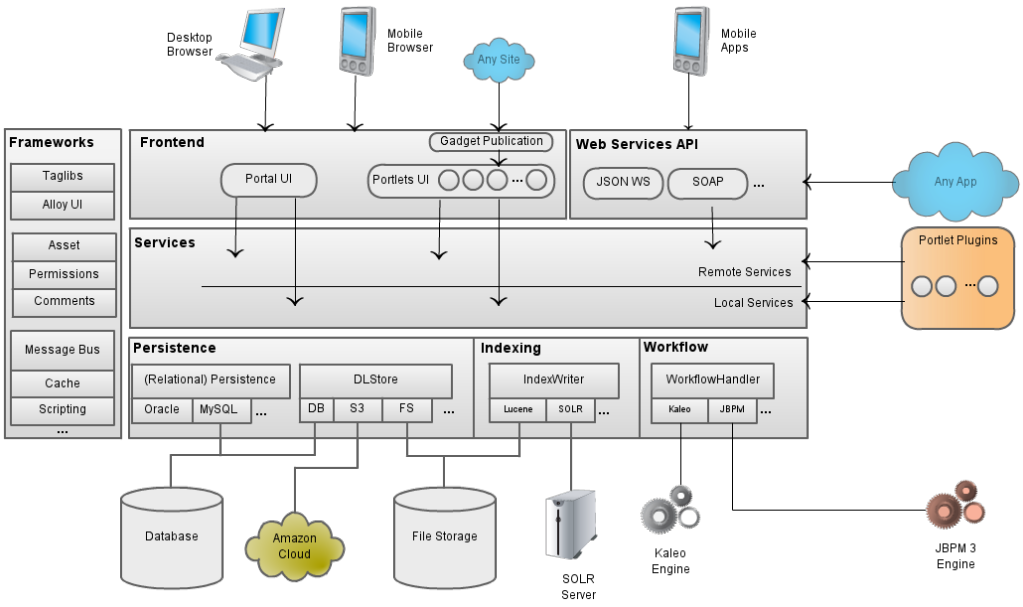

Figura 2.4: Architettura Liferay Portal [25].

In figura 2.4 viene presentata l'architettura di Liferay ed illustrati gli strati, descritti di seguito, di cui è composto:

Frontend layer questo livello definisce l'interfaccia dell'utente finale.

Service layer al suo interno sono contenute la grande maggioranza delle logiche di business, per la piattaforma del portale e per tutte le portlet incluse e pronte all'uso.

Persistance layer in questo strato troviamo Hibernate, la tecnologia principale attraverso il quale vengono eseguiti gli accessi al database.

Web service API layer a questo livello vengono gestiti i servizi web, come JSON e SOAP.

In Liferay, i tre strati che compongono il backend, ossia service, persistance e web service API, vengono generati in maniera automatica, il componente che si occupa di questa operazione viene chiamato Service Builder.

Il Service Builder è il collante che unisce questi strati e fornisce allo sviluppatore un livello di astrazione superiore, riducendo la complessità di utilizzo e mascherando lo stack tecnologico sottostante, tra queste tecnologie troviamo Spring ed Hibernate [26].

2.2.2 Municipium App

Municipium App è un progetto nato nel 2015, attualmente sono impegnati 9 sviluppatori sulle attività di manutenzione e ampliamento delle funzionalità del prodotto, questi sono suddivisi sulle sedi di Santarcangelo di Romagna e Cesena.
Nel 2019 è stata rivista in maniera importante l'architettura dell'applicativo, il che ha portato ad un prodotto molto più stabile ed appetibile per i clienti, tanto da portare nel 2020 ad un aumento del 60% dei comuni serviti [22].
Allo stato attuale conta più di 400 comuni attivati, i quali possono essere tutti facilmente consultati su un'unica applicazione, grazie alla gestione multi-ente su cui si fonda [27].

Municipium è un'app pensata per fornire alle amministrazioni locali una comunicazione semplice e diretta con cittadini e imprese. Fornisce alle amministrazioni un unico punto, attraverso un pannello di controllo dedicato, per gestire servizi ed informazioni sia dell'applicazione sia del sito comunale. Municipium è un applicativo modulare le cui funzionalità, riassunte in seguito, possono essere attivate a piacimento a seconda delle esigenze:

News ed Eventi possibilità di notificare notizie ed eventi geolocalizzati anche in maniera automatizzata.

Notifiche push grazie a questo tipo di messaggistica istantanea, è possibile adottare una comunicazione diretta ed immediata con i cittadini per comunicazioni importanti o emergenze.

Mappe interattive l'applicazione è predisposta all'uso di mappe interattive per comunicare in maniera più efficace con gli utenti.

Rifiuti smart un insieme di utilità a supporto della gestione dei rifiuti, come calendario porta-a-porta, promemoria e glossario del riciclo.

Segnalazioni possibilità per cittadini di inviare in maniera riservata e semplice segnalazioni all'amministrazione.

Sondaggi possibilità di inviare sondaggi ai cittadini per sondare le loro opinioni.

Protezione Civile comunicazione in tempo reale di rischi e stati di allerta in relazione al territorio.

Personalizzazione l'applicazione è configurabile in molti dei suoi aspetti per garantire un'adeguata personalizzazione.

Informazioni Utili possibilità di consultare le informazioni generali sull'amministrazione, il territorio e i servizi.

Multe smart possibilità per i cittadini di effettuare il pagamento delle multe, grazie all'integrazione con il software Concilia per la gestione delle violazioni al Codice della Strada.

Pillole informazioni di utilità di vario genere preparate per i cittadini.

Assistenza e statistiche l'applicazione è dotata di un pannello di controllo con istruzioni e cruscotti riepilogativi, inoltre viene offerto un servizio di assistenza online e telefonica sempre garantita.

2.2.3 Sportello telematico polifunzionale

Lo Sportello telematico polifunzionale affianca il sito istituzionale dell'amministrazione ed offre un servizio di presentazione pratiche al pari di quello offerto da J-City Gov, all'interno del quale viene gestito nel modulo presentato chiamato SGPI.
L'applicativo è stato progettato prestando la massima attenzione in materia giuridica alle leggi fondamentali che regolano il rapporto tra Pubblica

Amministrazione e cittadino. L'interfaccia del servizio è stata pensata per semplificare l'accesso ai servizi da parte dei cittadini, questo attraverso una cura del linguaggio ed un processo di sburocratizzazione delle procedure amministrative, tutto ciò nel massimo rispetto delle linee guida per i siti web delle pubbliche amministrazioni e delle norme sull'accessibilità.

Lo sportello telematico polifunzionale si basa sulla piattaforma Drupal. Drupal è una progetto open-source, scritto in linguaggio PHP e distribuito sotto licenza GNU GPL. É uno tra i software di Content Management System (CMS) più diffuso ed è la piattaforma che molte agenzie governative negli Stati Uniti, a Londra e in Francia usano per comunicare con i cittadini.
Drupal offre un insieme di funzionalità standard, come la facile creazione di contenuti e un'eccellente sicurezza, unite a prestazioni affidabili. A queste aggiunge una notevole flessibilità e fa della modularità uno dei suoi principi fondamentali, ciò consente di estendere i servizi base offerti dalla piattaforma con moduli e temi per soddisfare le esigenze richieste, inoltre è possibile contare anche su un'ampia gamma di moduli sviluppati dalla vasta comunità collegata al progetto [28].

2.3 Analisi dello stato attuale dei servizi

I prodotti descritti godono tutti di una discreta diffusione e sono tutti attivi sul mercato ormai da diversi anni. La loro longevità è dovuta non solo alla qualità e alla cura con il quale sono stati realizzati ma anche ai continui aggiornamenti e alle evoluzioni che hanno subito nel tempo.
L'attenzione per i clienti e la cura del prodotto, hanno portato ciascuno degli applicatavi a concentrarsi su particolari aspetti, in funzione dei riscontri ricevuti e dei desideri dei propri utenti. Tutto ciò ha permesso di offrire un servizio sempre aggiornato ed affidabile negli anni, garantendo la soddisfazione dei propri utenti con conseguente fidelizzazione.
Ognuno dei prodotti possiede delle funzionalità in comune agli altri appli-

cativi, in particolare J-City Gov e Sportello Telematico, ma ciascuno di essi è dotato anche di pregi e funzionalità di cui non godono gli altri prodotti. Queste sono le ragioni che hanno portato alla loro diffusione nel mercato e che li rendono non intercambiabili.

Attualmente, tutti gli applicativi sono quindi in manutenzione e su ciascuno di essi vi è l'esigenza di implementare nuove funzionalità. Molte delle richieste di sviluppo che giungono riguardano però servizi già offerti dagli altri prodotti, quando così non è, sono invece in comune a più di essi e devono quindi essere rilasciati su ciascuno.
Questa metodologia di lavoro non è affatto efficiente, inoltre la distribuzione sul territorio delle sedi del gruppo, imprescindibile viste le dimensioni dell'azienda, rende più complicato lo scambio di conoscenze e ostacola il confronto quando si sviluppano funzionalità simili o in collaborazione, impedendo ancora una volta una ottimizzazione degli sforzi erogati.

La volontà dell'azienda è quella di offrire sempre un servizio aggiornato e moderno con una particolare attenzione per la sicurezza. Per riuscire in questo intento è necessario essere al passo con le nuove tecnologie e gli standard emergenti.
Mantenere un software sempre aggiornato con l'ultima versione disponibile di ciascuna delle librerie e dei componenti che sfrutta, è un lavoro impegnativo e alle volte non sempre praticabile. Migrare per esempio J-City Gov dalla versione 6.2 di Liferay alla versione 7 richiederebbe uno sforzo eccessivo viste anche le dimensioni del progetto stesso, un aggiornamento del prodotto è però necessario e J-City Gov non è l'unico applicativo all'attivo da anni che trarrebbe molti vantaggi da un'opera di modernizzazione.

Un ulteriore fenomeno che coinvolge l'intero gruppo, è stato denunciato dai membri del team che lavora allo sviluppo dello sportello telematico, attualmente alle prese con l'integrazione dei servizi di back-office, richiesta in

maniera inderogabile dalla direzione a seguito dell'ingresso in azienda.
Il problema nasce dal fatto che i prodotti dell'area dei servizi al cittadino siano integrati con quasi la totalità degli applicativi aziendali, questo comporta il fatto che al proprio interno, il gruppo debba raccogliere tutte le competenze aziendali dei singoli domini. Queste sono numerose e complesse, pensiamo per esempio alle conoscenze necessarie per operare all'interno dei servizi tributari oppure a quelle collegate ai procedimenti edilizi.
La situazione è particolarmente delicata per i membri del team entrato a far parte dell'azienda solo di recente, in realtà però è un fenomeno che riguarda da sempre il gruppo, causando notevoli difficoltà durante la fase di onboarding del nuovo personale e la riassegnazione dei compiti tra i membri del team stesso.

Infine, bisogna considerare che il gruppo di servizi al cittadino si trova coinvolto all'interno del processo di evoluzione tecnologica aziendale avviato nel 2015. Inizialmente questo processo è partito con l'obiettivo di aggiornare la suite Sicr@web, in seguito è stato esteso ad altri prodotti aziendali.
Il primo passo è stato quello di aggiornare gli applicativi scritti in Java, portando alla versione 8 i software che ancora non avevano compiuto questa migrazione in precedenza.
Successivamente è stata rivista l'architettura di alcuni servizi tra cui Municipium App, come già citato all'interno della sezione 2.2.2, per adeguarli agli ambienti cloud ed istituire un processo di integrazione e distribuzione continua.
Inoltre, è stata intrapresa un'iniziativa volta a migliorare l'usabilità dei prodotti che ha portato successivamente alla riscrittura completa, tuttora in atto, dell'interfaccia utente di Sicr@web. Precedentemente era sviluppata esclusivamente in Java mediante l'uso della libreria Swing ma nel 2018 è stata avviata la formazione dei dipendenti, al termine della quale è cominciata la revisione delle prime parti dell'interfaccia e quindi la migrazione a tecnologie web, basate in particolare sul framework Angular.

Capitolo 3

Il processo di trasformazione

In questo capitolo vedremo l'approccio adottato dall'azienda per perseguire gli obiettivi prefissati, definiti a seguito delle necessità emerse in funzione di quello che è lo stato attuale del gruppo servizi al cittadino, presentato in sezione 2.3, e che possiamo riassume nell'esigenza di istituire un processo di trasformazione che porti dalla situazione attuale, caratterizzata da tre prodotti distinti con funzionalità parzialmente differenti, verso l'unificazione di questi e la creazione di un unico applicativo in grado di soddisfare le esigenze di tutti i clienti, attuali e futuri.

Verrà quindi analizzato il processo di trasformazione e definite le fasi che lo caratterizzano, per arrivare a stabilire un approccio generalizzato, applicabile, con i dovuti adattamenti, a qualsiasi contesto di questo tipo.
Verrà presentato il pattern *Strangler Fig Application* e quindi il principio alla base del processo, introdotto, al termine di un'analisi dei requisiti, per istituire un processo solido e rigoroso. In seguito, verranno analizzate le fasi individuate, studiati i singoli passaggi che le compongono e che hanno portato alla definizione della soluzione finale, basata su una architettura a micro-frontend.

Il caso di studio rappresenta dunque un'occasione, per analizzare e caratterizzare un fenomeno che investe qualsiasi prodotto al termine del suo

ciclo di vita, ovvero il raggiungimento dello stato di obsolescenza e la conseguente necessità per i produttori di migrare verso una nuova soluzione. Tutto ciò, all'interno di un complesso contesto contrassegnato da un coefficiente di difficoltà ulteriore, dovuto dalla necessità di integrare più prodotti contemporaneamente.

3.1 Necessità emerse

All'interno del capitolo 2 è stato analizzato il contesto aziendale e in particolare approfondita la struttura, i prodotti e la situazione del gruppo chiamato servizi al cittadino. Proprio in funzione della situazione attuale, analizzata in sezione 2.3, è possibile effettuare una serie di considerazioni, al fine di individuare ed approfondire quelle che sono le esigenze del gruppo, presenti e future.

La prima considerazione che si può fare è che attualmente ciascuno degli applicativi gode di un certo attaccamento maturato nel tempo e benché vi siano varie sovrapposizioni tra le funzionalità offerte, nessuno dei tre riesce a primeggiare in maniera assoluta sugli altri. Questo impedisce all'azienda di promuovere un unico prodotto tra quelli esistenti ed incentivare la migrazione dei clienti verso uno solo dei propri applicativi senza generare malcontenti.

Tra i prodotti in servizio Municipium App è emerso essere quello basato su tecnologie più moderne, questo anche in funzione della revisione architetturale subita in tempi recenti. Invece, J-City Gov richiederebbe un importante aggiornamento del framework, la versione 6.2 di Liferay è stata sostituita dalla 7 ormai da tempo e ciò espone l'applicativo a vulnerabilità di sicurezza sempre maggiori, passare alla versione successiva però pare essere un investimento troppo importante per le risorse del team.

Unita alla necessità di aggiornare i prodotti in essere, si palesa anche l'esigenza di convergere verso un'unica soluzione applicativa. Procedere mantenendo le sovrapposizioni attuali di funzionalità tra i vari software, consiste

in un uso non efficiente di tempo e risorse per l'azienda. Questo processo benché sia indispensabile e da attuare nel minor tempo possibile, deve avvenire anche in maniera sostenibile, in funzione delle disponibilità del gruppo servizi al cittadino.
Gli applicativi attualmente in uso devono poter continuare ad erogare i propri servizi durante tutto il processo, inoltre deve essere garantita anche la stabilità, la sicurezza e l'adeguatezza degli applicativi, esattamente come i clienti si aspettano, in linea anche con quelle che saranno le normative uscenti da parte del legislatore, le quali devono essere rispettate dalle amministrazioni locali.

In materia di norme da rispettare bisogna infatti considerare il settore in cui opera l'azienda, ovvero quello della Pubblica Amministrazione, in particolare dei servizi offerti direttamente ai cittadini. In questo contesto l'ente di riferimento è l'Agenzia per l'Italia Digitale[1] (AgID) che si occupa di emanare linee guida contenenti regole, standard e guide tecniche.
La finalità è quella di garantire ai cittadini la fruizione di siti istituzionali che rispettino i principi di accessibilità, nonché di elevata usabilità e reperibilità, anche da parte delle persone disabili, completezza di informazione, chiarezza di linguaggio, affidabilità, semplicità dì consultazione, qualità, omogeneità ed interoperabilità.

L'analisi effettuata ha inoltre evidenziato varie difficoltà dovute alla convergenza delle conoscenze dei vari domini aziendali in carico al gruppo. L'onore di acquisire e padroneggiare queste conoscenze, particolarmente vaste ed intricate in certi contesti, comporta una serie di ostacoli. Questo in particolare in fase di onboarding di nuovo personale e di riassegnazione dei membri dei team sui diversi moduli degli applicativi.
Al fine di far fronte a questo fenomeno si ritiene necessaria una riassegnazione delle competenze in funzione delle conoscenze, almeno in quei contesti caratterizzati da domini molto articolati. Tutto ciò, attraverso il disegno

[1] `https://www.agid.gov.it/`

di un processo che consenta al gruppo, dei servizi al cittadino, di ricevere un maggior supporto dagli altri reparti aziendali, per ottimizzare la gestione delle conoscenze e favorirne la condivisione.

Infine, non si deve trascurare il processo di innovazione aziendale in atto. Proseguire con le direttive aziendali, all'interno di quello che è il disegno generale di evoluzione, deve essere considerata una priorità.
In particolare, sarà necessario aggiornarsi adottando le tecnologie di riferimento nei casi in cui sia previsto il passaggio a nuove soluzioni. Inoltre, è previsto che le nuove architetture siano sviluppate su ambienti cloud ed è fortemente incentivata l'istituzione di processi di automazione, volti all'adozione di pratiche di integrazione e distribuzione continua.

3.2 Formalizzazione dei requisiti

Nella sezione precedente sono state individuate le esigenze ed i vincoli che dovranno essere tenuti in considerazione durante la definizione del processo di trasformazione, volto alla completa integrazione dei prodotti attualmente disponibili e all'ottimizzazione dell'uso delle risorse del gruppo, al fine di migliorare i servizi offerti.

A fronte delle considerazioni fatte possiamo riassumere come segue le necessità emerse:

- Dismissione di J-City Gov.
- Convergenza verso un'unica soluzione applicativa.
- Completo supporto a tutti i prodotti fino alla loro cessazione.
- Riassegnazione delle competenze in funzione delle conoscenze.
- Attuazione del processo di innovazione aziendale.
- Preservazione del rispetto delle linee guida, dei regolamenti e degli standard aziendali e nazionali.

In sintesi, sarà pertanto necessario evolvere verso un nuovo sistema, attuando un processo che consenta di fare ciò senza richiedere un cambiamento radicale ed un investimento eccessivo in termini economici e di risorse dispiegate, in maniera graduale per consentire al gruppo di continuare, anche se in maniera ridotta, le attività sui software attualmente in uso. Il tutto mirando alla creazione di un sistema basato sulle tecnologie più moderne, i massimi standard di sicurezza, una particolare attenzione per la qualità e la massima capacità di adattarsi alle evoluzioni future.
Durante la fase di pianificazione del processo sarà anche necessario tenere in considerazione le attuali competenze dei dipendenti e le loro attitudini, inoltre la distribuzione spaziale degli elementi del gruppo e quindi l'esigenza di garantirgli la dovuta indipendenza, senza sottovalutare le questioni tecniche collegate alla soluzione. Sarà infatti fondamentale giungere ad un sistema in grado di scalare in maniera rapida ed efficiente considerando anche le interazioni che il software dovrà mantenere con i gestionali aziendali di back-office. Questo andrà unito al processo di innovazione aziendale in atto, che mira ad una maggiore efficienza grazie all'integrazione con le principali tecniche in ambito integrazione e distribuzione continua (CI/CD). Pertanto sarà obbligatorio individuare un architettura che ben si presta all'implementazione di un processo di automazione rigoroso, in linea con quelle che sono le direttive da seguire.

3.3 Il principio alla base della trasformazione

L'analisi effettuata ha palesato l'esigenza di convergere verso un'unica soluzione applicativa. La necessità è quella di attuare un cambiamento graduale, senza effettuare una riscrittura completa del software, mantenendo in funzione ed aggiornati gli applicativi aziendali esistenti, al fine di non creare alcun tipo di disservizio e di agevolare il più possibile il passaggio ad un'unica soluzione software in maniera ordinata.

Per raggiungere questo obiettivo si è deciso di applicare il pattern *Stran-*

gler Fig Application. Questo principio è stato introdotto da Martin Fowler, il quale, durante un viaggio in Australia in visita alle foreste pluviali sulla costa del Queensland, ha preso ispirazione da una pianta chiamata Ficus watkinsiana, nota anche col nome comune di "fico strangolatore" [29].
I semi di queste piante, depositati da vari animali sulle fronde degli alberi più alti, germogliano velocemente grazie alla posizione che li espone alla luce del sole di cui necessitano. In questo modo sviluppano delle lunghe radici che nel tempo raggiungono il suolo, discendendo lungo il tronco della pianta che li ospita e formando vari intrecci. Nel corso di molti anni crescono notevolmente dando vita a forme spettacolari, fino al punto di soffocare la pianta ospite privandola della luce e delle sostanze nutritive.
Questo comportamento ha ispirato Fowler nell'ideare una metodologia per riscrivere sistemi complessi di notevoli dimensioni. Il concetto consiste nel provvedere ad una riscrittura senza partire da zero, bensì appoggiandosi al sistema esistente, emulando quindi il comportamento del fico strangolatore [29].

Questo approccio si basa sull'idea di generare valore in maniera incrementale per l'azienda e per l'utente, rilasciando gradualmente parti del prodotto finale, invece di sospendere gli aggiornamenti per lungo tempo, per poi rilasciare una nuova soluzione applicativa. In questo modo, è possibile fornire valore in modo costante, mentre effettuare rilasci frequenti consente di monitorare i progressi con maggiore attenzione e di individuare il percorso migliore da seguire in base ai riscontri ricevuti. L'investimento iniziale per il team di sviluppo è piuttosto moderato e può generare immediatamente benefici per l'utente finale [12].
Per queste ragioni, questo approccio è stato largamente impiegato per il passaggio da sistemi monolitici ad architetture a microservizi [30], la sua adozione è stata promossa anche da IBM [31] e Microsoft [32].

Il principio da adottare sarà quello di sviluppare nuovi componenti esportabili, i quali andranno a rimpiazzare le parti esistenti degli applicativi in

attività e ad integrare nuove funzionalità in quelli che ne erano sprovvisti. Questo fino a quando le funzionalità offerte dagli applicativi non saranno di pari livello e quindi sarà possibile migrare i clienti verso un unico prodotto.
Per individuare le funzionalità da racchiudere nei singoli componenti, verrà analizzato l'ambito e i confini del dominio applicativo, adottando quindi un approccio DDD, in modo da realizzare componenti caratterizzati da un contesto e delle responsabilità ben definite.
Questo modello ovviamente presenta un insieme di sfide da superare, infatti durante il processo sarà necessario apportare le dovute modifiche agli applicativi, in modo da poter integrare efficacemente i componenti sviluppati.

3.4 I passaggi della trasformazione

Nella sezione precedente è stato illustrato il principio che dovrà essere adottato per raggiungere lo stato desiderato e completare quindi il processo di integrazione ed evoluzione dei servizi al cittadino, cominciato con l'acquisizione dell'azienda GLOBO. Nel seguito, vedremo come questo principio verrà applicato, per definire quelli che sono i passaggi che porteranno a raggiungere gli obiettivi attesi.

Le fasi individuate come le principali componenti del processo di trasformazione sono due. La prima consiste in una fase di integrazione, in cui gli applicativi esistenti dovranno essere mantenuti in attività ed aggiornati, mentre l'obiettivo da raggiungere per i vari team sarà quello di integrare tra loro i prodotti in possesso.
In questa fase l'azienda continuerà a proporre, ai nuovi clienti, tutti e tre i software a disposizione. Questa manovra commerciale, nasce dall'esigenza di coprire il più possibile le domande del mercato, visto che ognuno dei prodotti possiede funzionalità specifiche. Proseguire con la vendita consentirà di raggiungere il maggior numero di clienti, per questo motivo, agli interessati, potranno essere proposte soluzioni che prevedono l'utilizzo contemporaneo di prodotti diversi del gruppo.

Vista la decisione presa, di proporre tutte le soluzioni applicative a disposizione, è necessario fare in modo che tutti i prodotti siano in grado di trasmettere una visione comune e coerente dei servizi e quindi che subiscano un processo di uniformazione.
Per ottenere questo risultato, è fondamentale che, tutti e tre i team del gruppo dei servizi al cittadino, sposino i valori e i principi dell'azienda ed in particolare lavorino per garantire una certa omogeneità di stile.

Raggiunto questo obiettivo, sarà possibile procedere con la seconda parte della fase di integrazione. Questa consiste nel passare ad una nuova strategia di sviluppo che porterà all'implementazione di nuovi servizi, sotto forma di componenti, integrabili su tutti gli applicativi.
Le funzionalità già esistenti invece, verranno rifattorizzate e riscritte a loro volta come componenti importabili. Questi andranno in seguito a sostituire i moduli precedenti dei prodotti che già fornivano quel determinato servizio e ad ampliare il catalogo di quelli che ne erano sprovvisti.

Una volta che i vari prodotti avranno superato questa fase di integrazione e raggiunto un livello di maturazione adeguato, sarà possibile procedere con la fase successiva, ovvero quella di migrazione.
Al termine del processo di integrazione infatti, i prodotti aziendali avranno appianato il dislivello esistente, in termini di funzionalità offerte, che attualmente li contraddistingue.
In un contesto di questo genere il passaggio dal prodotto in uso verso una nuova soluzione, potrà essere vissuto dagli utenti in maniera adeguata, evitando che si generi in essi una forma di avversione al cambiamento che metta in pericolo la riuscita del progetto. L'obiettivo sarà invece, quello di creare le condizioni per il quale il cambiamento possa essere percepito in maniera positiva, grazie al fatto che le funzionalità offerte in precedenza saranno mantenute ma proposte mediante una soluzione moderna e più sicura. In questo modo, sarà possibile commercializzare un unico prodotto in grado di rimpiazzare le soluzioni attuali.

Le fasi appena descritte, che caratterizzano il processo di trasformazione, e i passaggi che le compongono, possono quindi essere riassunte in maniera sintetica, come segue:

1. Integrazione:
    - Uniformazione dei prodotti.
    - Condivisione dei servizi.
2. Migrazione:
    - Adozione di un'unica soluzione software.

Nelle sezioni seguenti verranno approfonditi i passaggi che compongono il processo di trasformazione ed illustrato con maggiore dettaglio gli approcci e le tecniche che il gruppo intende adottare.

3.4.1 Uniformazione dei prodotti

Questo primo passaggio della fase di integrazione, è stato avviato nel 2020, contestualmente all'incorporazione dell'azienda GLOBO. Infatti, a seguito dell'annuncio dell'acquisizione sono cominciati i lavori per integrare i nuovi colleghi all'interno del gruppo di servizi al cittadino, introducendo quindi un nuovo team e il relativo prodotto, Sportello Telematico, come descritto in sezione 2.2.
Sempre a partire da questo momento, al marchio GLOBO è stato affiancato quello Maggioli, sia nelle comunicazioni al pubblico sia all'interno dei prodotti, questo anche ad indicare l'unione con l'identità del brand aziendale e la condivisione dei suoi valori e dei suoi principi.
Al termine dell'acquisizione, il marchio GLOBO, non ha però cessato di esistere in quanto rappresenta un valore importante e nutre di grande stima e considerazione da parte di molti suoi affezionati clienti.

Come anticipato, questa fase è stata caratterizzata dalla commercializzazione di tutti e tre gli applicativi, inclusa ovviamente la manutenzione e la loro evoluzione. Attualmente, a seguito della conclusione di questo periodo, si sta incentivando maggiormente la commercializzazione di tutti i prodotti anche presso i clienti che già sono in possesso di uno di questi. Ciò è dovuto alle attività svolte per omogeneizzare e quindi uniformare gli applicativi.

Tra gli sviluppi svolti, è importante far notare che è stato realizzato un servizio di Single Sign-On (SSO), per consentire agli utenti di accedere a tutti i prodotti del gruppo con le stesse credenziali, una singola volta.
Altre attività sono state svolte in merito all'omogeneizzazione dello stile che, come anticipato in sezione 3.1, è fortemente influenzato dalle linee guida definite dall'Agenzia per l'Italia Digitale. AgID però, si è impegnata anche per fornire un aiuto concreto, questo attraverso il progetto Designers Italia[2] e il suo design UI Kit[3], ha infatti supportato la creazione di alcune librerie open-source di ausilio per lo sviluppo di interfacce e il mantenimento di un *design system* solido e coerente:

Bootstrap Italia libreria open-source e principale punto di riferimento per la costruzione di interfacce per servizi della PA. Costruita sulle basi dello UI Kit e sulla libreria Bootstrap[4] 4.5, da cui eredita componenti, griglie e classi di utilità, personalizzandole secondo le linee guida di design per i siti web della Pubblica Amministrazione.

Angular Kit componenti sviluppati in linguaggio JavaScript, costruiti sulle basi di Angular[5] 6. La libreria è ancora in fase di sviluppo e non contiene tutti i componenti disponibili nello UI Kit o in Bootstrap Italia, non riceve però aggiornamenti ormai da tempo.

[2] `https://designers.italia.it/`
[3] `https://github.com/italia/design-ui-kit`
[4] `https://getbootstrap.com/`
[5] `https://angular.io/`

React Kit componenti sviluppati in linguaggio JavaScript, costruiti sulle basi di React[6].

Web Toolkit libreria deprecata sostituita da Bootstrap Italia, la quale è stata aggiornata con le nuove direttive introdotte nella più recente versione dello UI Kit, permettendo di semplificare lo sviluppo di siti web della Pubblica Amministrazione conformi con le nuove linee guida di design.

In questo contesto, Municipium App è stata costruita sulle basi di Bootstrap Italia mentre, il progetto J-City Gov, è stato integrato da tempo con la medesima libreria per quasi la totalità dei servizi offerti. Invece, Sportello Telematico è basato sul Web Toolkit, visto che la libreria è stata deprecata il passaggio al suo successore era già in piano e il desiderio di omogeneizzare gli applicativi rappresenta un motivo ulteriore per accelerare questo processo.

3.4.2 Condivisione dei servizi

Questa è la fase in cui si trova il progetto allo stato attuale dei lavori, come anticipato, è caratterizzata dalla commercializzazione di tutti e tre gli applicativi. In questo senso è possibile affermare che, a seguito dei lavori effettuati in precedenza, la presenza simultanea di diversi prodotti presso uno stesso cliente, comporta un valore aggiunto rispetto alla mera somma delle parti, ciò grazie alle attività svolte per omogeneizzare e quindi uniformare gli applicativi.
La manutenzione e l'aggiornamento di tutti i prodotti è ancora in corso, ma lo sviluppo di nuove funzionalità deve avvenire secondo il principio di condivisione annunciato.

Giunti a questo punto del processo di trasformazione è possibile cominciare a mettere in pratica quanto descritto in sezione 3.3, passando ad una metodologia di lavoro che prevede lo sviluppo di nuovi servizi sotto forma

[6] `https://it.reactjs.org/`

di componenti esportabili. Questi dovranno sostituire i moduli applicativi che fino a quel momento svolgevano il medesimo compito, invece, una volta integrati nei prodotti sprovvisti, ne arricchiranno il valore con nuove funzionalità.

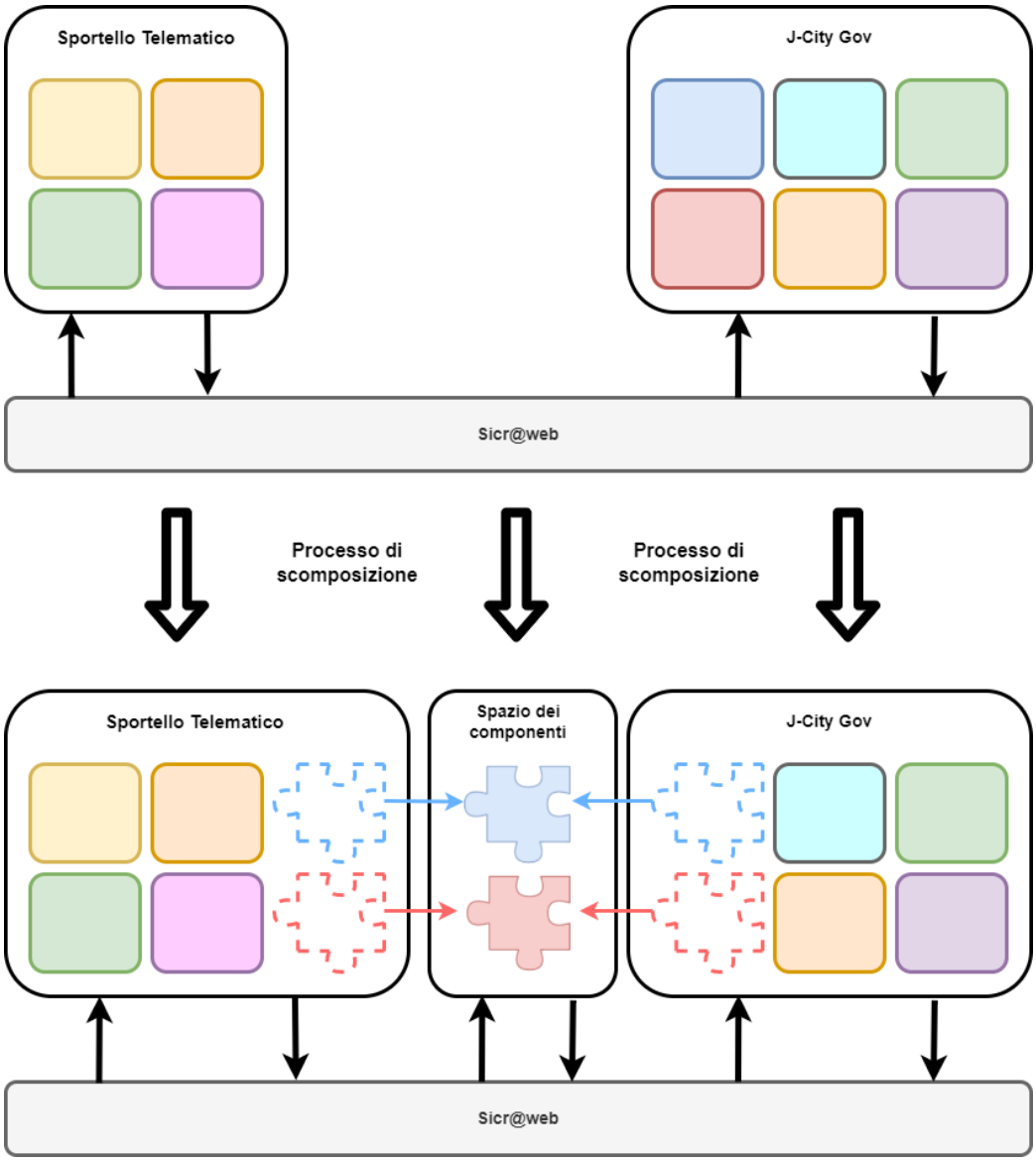

Figura 3.1: Scomposizione dei moduli applicativi in componenti esportabili e condivisi tra i prodotti.

Durante l'analisi delle attività da svolgere in questa fase, è stato fondamentale tenere bene a mente due fattori, isolamento ed autonomia. Garantire queste due proprietà ai componenti prodotti e al loro processo di sviluppo, sarà infatti indispensabile per il conseguimento del successo, principalmente per due ragioni.
Il primo motivo è dovuto dalla conformazione attuale del gruppo dei servizi al cittadino, presentata in sezione 2.2. Infatti, il gruppo non solo è composto da tre diversi team distinti ma questi sono anche suddivisi su quattro diverse sedi aziendali. Questo fattore richiede che le attività possano essere svolte in maniera indipendente, riducendo al minimo la necessità di coordinamento tra i diversi team.
La seconda ragione per cui si rende necessario raggiungere un alto livello di autonomia ed isolamento, è data dal fatto che alcuni componenti saranno poi assegnati in gestione ai membri dei team di Sicr@web, specializzati nel relativo ambito applicativo. Consentendo di adottare una piena gestione full-stack dei team di sviluppo, al fine di garantire il pieno controllo dello specifico dominio assegnato.
Questo solo per quei componenti che possiedono effettivamente un modulo corrispettivo lato back-office ed almeno per quelli che si contraddistinguono per la complessità e la specificità dell'ambito in cui operano.
Il tutto nasce per rispondere al requisito emerso, ossia la riassegnazione delle competenze in funzione delle conoscenze, nato dalla difficoltà di gestione dovuta alla convergenza delle conoscenze dei vari domini aziendali in carico alla divisione dei servizi al cittadino.

La scomposizione degli applicativi in più servizi, avverrà adottando l'approccio Domain-Driven Design (DDD). Quindi, analizzando il dominio di ogni applicativo, verranno individuati i vari sottodomini in cui può essere scomposto, così si andranno a delineare i relativi *buonded context*. Ossia, i confini dei singoli contesti logici, dichiarati in maniera delimitata e definita che dovranno nascondere i dettagli implementativi attraverso un'interfaccia pubblica chiara e completa.

A partire dallo studio dei *bounded context* si organizzerà una architettura con diversi servizi, altamente disaccoppiati e che lavorano in modo collaborativo secondo il principio di singola responsabilità.
L'intenzione del gruppo è quella di ritardare la decisione finale sull'organizzazione dei singoli servizi fino all'ultimo momento utile e di procedere in maniera graduale, al fine di evitare una decomposizione prematura. In questo modo, vi sarà la possibilità di raccogliere il maggior numero di informazioni possibile e quindi sarà più facile individuare la direzione giusta da seguire.

A fronte delle considerazioni fatte in fase di definizione dei requisiti, saranno proprio i servizi offerti in J-City Gov i primi a subire questa trasformazione, in modo da anticipare il più possibile il pensionamento dell'applicativo. Questo processo, applicato al progetto J-City Gov, sarà agevolato dalla sua natura modulare. In fase di rifattorizzazione di ogni modulo, i principi enunciati verranno applicati al fine di valutare la necessità di introdurre una scomposizione ulteriore in più componenti, ciò anche per agevolare il processo di trasformazione generale.

La gestione a servizi isolati e la necessità di definire componenti autonomi uniti alle esigenze individuate in fase di analisi dei requisiti, ovvero di realizzare un sistema altamente scalabile, caratterizzato da uno sviluppo e una gestione in maniera agile ad opera di team interfunzionali indipendenti, con un'ottima gestione dei guasti e in grado di velocizzare le fasi di onboarding, rendono naturale convergere verso l'adozione di una architettura basata su microservizi e micro-frontend.
Queste architetture, ulteriormente sono anche in grado di fornire un supporto per diverse tecnologie e quindi di favorire l'integrazione tra i diversi prodotti, inoltre si sposano perfettamente con il desiderio aziendale di migrare verso soluzioni basate su ambienti cloud e si prestano in maniera eccellente all'adozione di processi di automazione.

3.4.3 Adozione di un'unica soluzione software

Una volta che i servizi offerti da J-City Gov saranno stati scomposti, incapsulati all'interno di componenti esportabili e condivisi con gli altri due prodotti, la fase di integrazione potrà dirsi ormai giunta al termine.
A questo stadio del processo di trasformazione i prodotti aziendali avranno quasi appianato il dislivello esistente, molte delle funzionalità offerte saranno incluse all'interno dei componenti comuni ed il ruolo degli applicativi sarà quasi esclusivamente limitato all'organizzazione dei micro-frontend.

In sostanza, i vari team responsabili dei singoli micro-frontend, potranno lavorare su ciascuna delle singole funzionalità in maniera indipendente in relazione al loro specifico dominio, la presentazione dei servizi però avverrà comunque in maniera coesa e coerente all'interno della specifica applicazione web, definita applicazione container o shell.
Questo è appunto l'elemento del sistema incaricato di orchestrare i singoli micro-frontend [33]. In modo simile a come ogni microservizio controlla il proprio database, ogni micro-frontend controlla solo il Document Object Model (DOM) del browser di cui è proprietario. L'applicazione shell supervisiona il DOM come risorsa condivisa e facilita la gestione del ciclo di vita di ciascun micro-frontend in fase di esecuzione [34].
Il principale interesse dovrà essere quello di cercare di mantenere questo elemento il più leggero e disaccoppiato possibile dal resto dell'applicazione, in modo che tutti i micro-frontend possano essere indipendenti ed evolversi senza alcuna subordinazione rispetto all'applicazione container [12].

Tipicamente i compiti principali assunti dall'applicazione container riguardano (*i*) la composizione dei micro-frontend all'interno dell'interfaccia utente, assieme agli elementi comuni come intestazione e piè di pagina; (*ii*) la gestione dell'instradamento dell'applicazione; (*iii*) il recupero e il caricamento in pagina dei micro-frontend quando necessario; (*iv*) la capacità di mettere a disposizione e di facilitare l'uso di strumenti comuni e risorse necessari a più componenti, come canali di comunicazione e dati condivisibili.

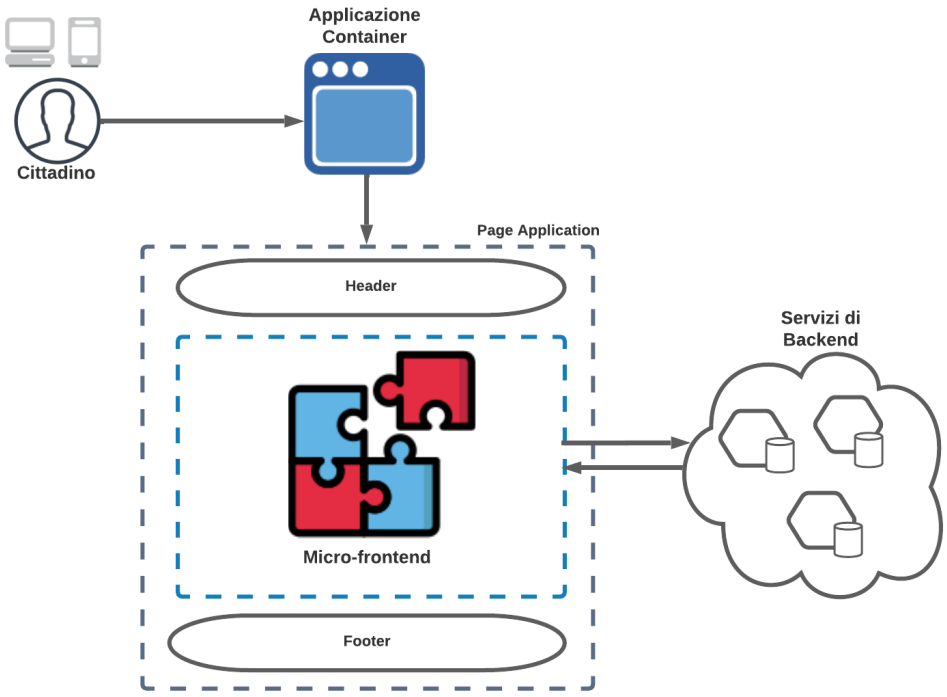

Figura 3.2: Schema di un'applicazione container usata come unica soluzione software.

Il compito di applicazione container potrebbe essere affidato ad uno degli applicativi esistenti, J-City Gov è però da scartare per i motivi noti, Sportello Telematico invece potrebbe essere una valida soluzione ma in questo caso Municipium è attualmente la scelta più quotata. Esso infatti, è in grado di fornire le maggiori garanzie in termini di longevità, visto che negli ultimi anni vi sono state varie rifattorizzazioni dell'applicativo che l'hanno reso un prodotto stabile ed aggiornato.
In alternativa, si potrebbe adottare una gestione che prediliga lo sviluppo di una nuova SPA per ricoprire il ruolo di applicazione shell. Gli utenti si aspettano che i siti web siano veloci e reagiscano prontamente, non voglio aspettare che la pagina venga ricaricata completamente a seguito di operazioni banali. Le persone tendono a preferire siti che reagiscono rapidamente,

pertanto è diventata sempre più popolare la diffusione di framework lato client come React, Angular e Vue.js [14]. Per questo motivo realizzare un nuovo sistema basato su una di queste tecnologie potrebbe essere una valida soluzione.

Attualmente non è ancora stata presa una decisione definitiva, molto dipenderà dall'evolversi della situazione con il proseguirsi degli sviluppi. Durante la prima fase del processo verranno analizzate in maniera approfondita le prestazioni degli applicativi, oltre che la loro efficacia nello svolgere questi compiti.
La scelta finale verrà quindi ufficializzata a ridosso dell'inizio di questa fase, a seguito della maggiore consapevolezza acquisita e all'evolversi del contesto tecnologico.

3.5 Definizione dell'approccio a micro-frontend adottato

In sezione 1.4, è stato presentato un insieme di tematiche che è utile discutere quando si approccia una architettura a micro-frontend, in particolare analizzando gli argomenti in relazione al contesto di utilizzo e ai requisiti specifici. Questo, al fine di adottare nel migliore dei modi possibili questo approccio, esaltandone i vantaggi conseguibili ed evidenziando gli svantaggi da gestire per prepararsi al meglio.

3.5.1 Aspetti principali

Verranno ora presentate le valutazioni e le scelte fatte, in relazione agli aspetti principali trattati in precedenza, applicati al contesto di studio corrente:

Ripartizione la suddivisione del frontend che si è deciso di attuare consiste in una ripartizione verticale che, come definito, è caratterizzata dalla

presenza di un unico micro-frontend in pagina per volta.
Questa è una diretta conseguenza delle necessità del gruppo dei servizi al cittadino, che vedono nell'adozione di una architettura di questo tipo, la soluzione all'esigenza di una decomposizione dei prodotti in funzione del dominio aziendale e non nel vantaggio di una ripartizione funzionale in componenti, finalizzata al riuso all'interno dello stesso applicativo.

Isolamento in sezione 3.4.2, sono state illustrate le ragioni per cui si rende necessario raggiungere un alto livello di autonomia ed isolamento, queste riguardano principalmente la conformazione del gruppo dei servizi al cittadino e l'esigenza di riassegnare le competenze in funzione delle conoscenze, sfociata nell'adozione di una organizzazione in team interfunzionali indipendenti con un livello di coordinamento richiesto minimo.
Inoltre, la necessità di integrare i micro-frontend all'interno di prodotti differenti, richiede l'utilizzo del giusto livello di incapsulamento per evitare interazioni non volute.

Comunicazione all'interno della sezione 1.4.2, sono stati approfonditi vari metodi di comunicazione che possono essere adottati con questo tipo di architettura. Le tecniche di comunicazione analizzate non sono mutuamente esclusive, pertanto non è conveniente legarsi ad una singola metodologia. Per questo motivo, a fronte delle esperienze acquisite in materia e alla condivisione di queste conoscenze all'interno del gruppo, verranno impiegate diverse soluzioni in base alle esigenze, alla situazione e al tipo di dato da scambiare.
Detto questo, tra le strategie di comunicazione lato client, si predilige l'adozione di uno scambio di informazione basato su Custom Event piuttosto che su Broadcast Channel API, visto il limitato supporto di quest'ultimo su Safari[7].

[7] `https://developer.mozilla.org/en-US/docs/Web/API/Broadcast_Channel_API#browser_compatibility`

Con estremo criterio, invece, verrà fatto uso di strutture comuni di comunicazione anche se ritenute estremamente pratiche. Inoltre, verranno adottate preferibilmente solo per la condivisione di informazioni non connotate da una semantica di dominio specifica ma più di carattere generale e possibilmente solo in maniera unidirezionale. Facendo sempre grande attenzione a non instaurare una forma di stato condiviso tra i micro-frontend, onde evitare un accoppiamento forte tra di essi.
In generale, la scelta fatta di adottare una suddivisione verticale agevola rispetto ad una orizzontale, dove l'esigenza di una strategia di comunicazione tra micro-frontend organizzata è molto più sentita.

Routing su questo argomento la scelta è stata fortemente condizionata dal contesto in cui ci troviamo, infatti la logica di routing è gestita direttamente dai tre specifici applicativi in maniera vincolante. I componenti che possiederanno più viste differenti, dovranno implementare direttamente in fase di caricamento le logiche per gestire correttamente i contenuti, ciò in base alla modalità in cui sono stati inclusi in pagina.

Coerenza della UX questo è un tema di primaria importanza per il gruppo, per questo motivo sono state svolte varie sedute di analisi in merito e attualmente non è ancora stata presa una decisione definitiva sull'approccio da adottare.
L'idea generale consiste nell'affidarsi ad una *design library* completa e affidabile, questo per via della dimensione del progetto, del numero di team che dovranno lavorarci sopra e delle difficoltà di comunicazione tra di essi, dovute alla gerarchia aziendale e alla distribuzione spaziale delle varie sedi.
Sulla decisione della libreria di componenti da utilizzare vi sono più alternative tuttora sotto esame, il requisito è quello di rispettare le linee guida fornite da AgID, come descritto in sezione 3.4.1. Una delle soluzioni potrebbe quindi essere quella di adottare direttamente una delle librerie messe a disposizione da designers italia.

In alternativa, si potrebbe sviluppare una libreria che includa una delle precedenti al suo interno, per esempio Bootstrap Italia, e fornisca una differente interfaccia di utilizzo per i componenti forniti dalla stessa.
Infine, si potrebbe decidere di sviluppare qualcosa di completamente personalizzato oppure basato su altri framework, al giorno d'oggi molto diffusi, come per esempio Twitter Bootstrap. Il tutto impegnandosi a prendere sempre le dovute precauzioni in materia di stile per rispettare le linee guida imposte.

Gestione delle dipendenze il fatto di aver deciso di adottare una suddivisione verticale dei micro-frontend comporta in generale meno rischi di duplicazione delle dipendenze.
Il fenomeno non deve tuttavia essere sottovalutato, le scelte future in materia di framework e librerie dovranno tenere conto attentamente di questo fattore, per questo andranno studiate le soluzioni migliori per ottimizzare gli applicativi e non incorrere in un tempo di caricamento iniziale della pagina eccessivamente elevato.

Composizione in materia di integrazione dei componenti sono state effettuate varie analisi, le motivazione che hanno portato alla decisione presa sono riportate in estrema sintesi nel seguito.
Un approccio basato su composizione in fase di compilazione è stato scartato, questo per garantire maggiore indipendenza anche in fase di rilascio, requisito fondamentale del progetto.
Un'integrazione tramite JavaScript è stata invece esclusa visto il ridotto livello di isolamento che questa gestione comporta, vantaggio che può essere ottenuto da una soluzione basata su iframe oppure su Web Component, di questi il secondo è stato preferito per la sua maggiore flessibilità rispetto al primo.
Successivamente, in sezione 3.5.3, verranno approfonditi i vantaggi di questo approccio e le sue caratteristiche tecniche, per fornire una visione completa della scelta.

3.5.2 Comunicazione tra micro-frontend e backend

In sezione 1.4.3, è stato introdotto un pattern di comunicazione frequentemente utilizzato in questo genere di architetture, noto con il nome di Backends For Frontends. All'interno del progetto si è deciso di predisporre un BFF per ogni micro-frontend che verrà sviluppato, la cui gestione sarà in carico al medesimo team.
I vantaggi conseguenti dall'adozione di questo pattern sono già stati discussi in precedenza, la riduzione di complessità, la maggiore resilienza e scalabilità che comporta, sono tutti fattori molto utili all'interno del contesto in esame ma il principale beneficio per cui si rende utile adottare questa soluzione, riguarda la netta separazione che è in grado di introdurre tra frontend e API server.

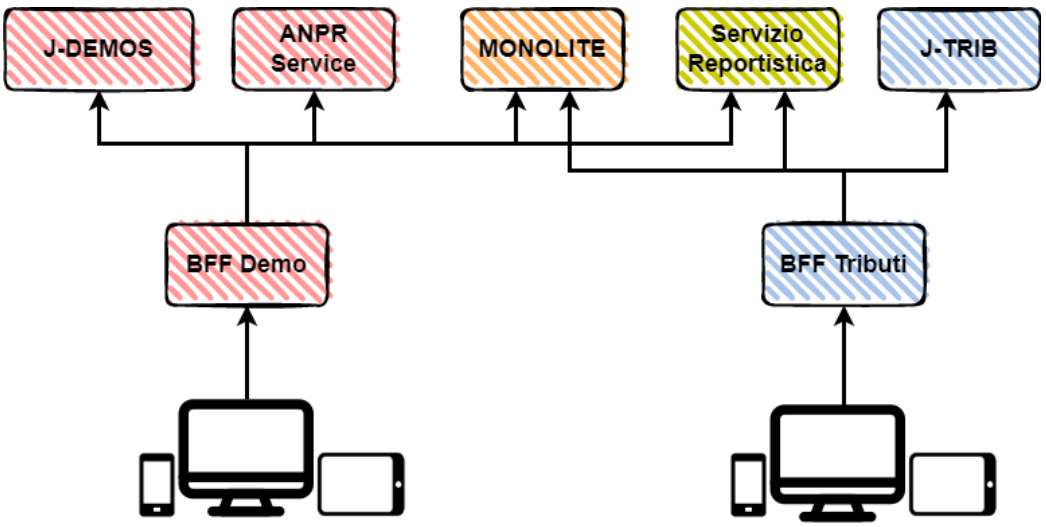

Figura 3.3: Applicazione del pattern BFF in relazione al contesto corrente.

Il motivo per cui si ritiene necessario introdurre questa separazione, riguarda il processo di evoluzione aziendale di cui è già stato a lungo discusso nelle sezioni precedenti. In particolare si fa riferimento alle iniziative di rinnovamento che stanno investendo Sicr@web, infatti è attualmente in corso un

processo che sta portando al rifacimento dei servizi esposti dall'applicativo. Le interfacce pubbliche di tipo SOAP di questi servizi stanno subendo una conversione verso un più moderno approccio basato su microservizi con API REST.

Durante lo sviluppo dei micro-frontend verrà quindi adottato il pattern BFF, anche in questo contesto congiuntamente ai principi definiti in sezione 3.3, caratteristici di un approccio basato sul modello *Strangler Fig Application*. Questo permetterà di nascondere le complessità conseguenti da questa migrazione dei servizi, semplificando lo sviluppo dei componenti frontend e svincolandoli dalle possibili modifiche future che questi potrebbero subire in funzione del cambiamento graduale in corso.

3.5.3 Lo strumento di composizione

Attualmente, svariati framework propongono un proprio sistema basato su componenti, adottare uno di questi non è però praticabile all'interno del nostro progetto. In quanto, comporterebbe vincolare il sistema ad un singolo ciclo di rilascio centralizzato, inoltre una modifica del framework richiederebbe una riscrittura completa.
I Web Component introducono invece un modello di componenti neutro e standardizzato indipendente dalla tecnologia, consentendo la coesistenza di applicazioni frontend isolate all'interno di una stessa pagina, anche quando il loro stack tecnologico non è lo stesso.

Per realizzare i micro-frontend del gruppo si è deciso quindi di adottare una strategia di composizione basata su Web Component. I motivi principali che hanno portato verso questa decisione, riguardano il fatto che essi forniscono una valida soluzione per incapsulare i diversi componenti in maniera standard, fornendo allo stesso tempo un modo per isolarli efficacemente, utile ad introdurli in sicurezza all'interno di applicativi legacy. Inoltre, consentono di essere sviluppati dai vari team lavorando in completa autonomia e quindi di rilasciare aggiornamenti in maniera indipendente.

Web Component è uno standard HTML emesso dal World Wide Web Consortium (W3C) [17], fornisce una soluzione nativa all'interno dei browser per realizzare elementi personalizzati, in cui regole di stile e codice JavaScript sono incapsulati al loro interno e le interazioni con il resto della pagina possono essere progettate in maniera agile, evitando comportamenti inattesi.

Web Component è una suite composta da diverse tecnologie:

Custom Element insieme di API JavaScript che consentono di definire elementi personalizzati e il loro comportamento, la definizione è incorporata nella specifica HTML del W3C e supportata in modo nativo in HTML 5.3 [35].

Shadow DOM pratico strumento per isolare regole di stile, definito all'interno della specifica DOM Living Standard[8]. Shadow DOM può essere visto come un sotto-albero incapsulato all'interno di uno specifico elemento.

HTML Template consentono di definire dei modelli html non mostrati al caricamento iniziale della pagina ma archiviati fino a quando non vengono istanziati tramite JavaScript. Questa tecnologia è definita all'interno dello standard HTML[9].

HTML Module successore dei deprecati HTML Import[10], il loro supporto consentirà di includere e riutilizzare documenti HTML come dipendenze. La standardizzazione sarà ad opera del Web Hypertext Application Technology Working Group (WHATWG) [17].

In vari contesti capita che venga fatto riferimento anche ad una sola delle singole tecnologie con il termine più generale di Web Component, ciò è dovuto

[8] `https://dom.spec.whatwg.org`

[9] `https://html.spec.whatwg.org/multipage/scripting.html#the-template-element`

[10] `https://www.w3.org/TR/html-imports/`

al fatto che ciascuno di questi standard può essere impiegato in maniera indipendentemente o in combinazione con un qualsiasi sottoinsieme degli altri [36].

L'approccio di base per l'implementazione di un Web Component parte dalla sua definizione. Per fare ciò è necessario creare una classe, attraverso la sintassi definita in ECMAScript 2015, in cui specificare le sue funzionalità. Estendere la classe `HTMLElement` è necessario per fare in modo che il tag, del Custom Element definito, possa essere regolarmente registrato dal browser e quindi interpretato correttamente all'occorrenza. Al fine di registrare l'elemento è necessario invocare il metodo `customElements.define(name, constructor, options)` definito dall'interfaccia `CustomElementRegistry`, nel listato 3.1 un esempio di quanto appena descritto.

```
1 class MyFirstComponent extends HTMLElement {
2   connectedCallback() {
3     this.innerHTML = '<h1>Hello World!</h1>';
4   }
5 }
6
7 customElements.define('my-first-component',
    MyFirstComponent);
```

Listato 3.1: Esempio di dichiarazione di un generico Custom Element.

Il nome del componente può essere scelto liberamente, l'unico vincolo da rispettare è che contenga almeno un carattere hyphen (-), in questo modo è possibile evitare problemi futuri, quando alla specifica HTML verranno aggiunti nuovi elementi [14].

I Custom Element, per la gestione degli eventi collegati al normale ciclo di vita, introducono un set di metodi che, se implementati, permettono di gestire gli eventi ad esso collegati tramite un'interfaccia standardizzata. I metodi appartenenti al ciclo di vita dei Custom Element sono:

- `connectedCallback`;
- `disconnectedCallback`;
- `adoptedCallback`;
- `attributeChangedCallback`;
- `formAssociatedCallback`;
- `formDisabledCallback`;
- `formResetCallback`;
- `formStateRestoreCallback`;

Il metodo `connectedCallback` è stato già mostrato nel listato 3.1, questo viene richiamato quando l'elemento viene aggiunto al DOM. Il suo opposto è `disconnectedCallback`, invocato una volta che l'elemento viene rimosso e che consente di gestire la fase di rilascio delle risorse. Invece, nel caso in cui venga aggiornato uno degli attributi verrà richiamato il metodo `attributeChangedCallback`.

In precedenza è stato introdotto lo Shadow DOM e discusso della sua capacità di isolare componenti. Per ottenere ciò, è sufficiente invocare il metodo `attachShadow` su un qualsiasi elemento HTML per dichiaralo come *shadow root*, ovvero come nodo di partenza appartenente al documento nel quale verrà incapsulato il sotto-albero, chiamato *shadow tree*, isolato dal resto della pagina, al quale spesso si fa riferimento con il termine *Light DOM*. In base a quanto appena descritto, l'esempio precedente può essere rivisto come illustrato nel listato 3.2.

```
1 connectedCallback() {
2     this.attachShadow({ mode: 'open' });
3     this.shadowRoot.innerHTML = `
4       <style>
```

```
5           h1 { color: blue; }
6         </style>
7         <h1>Hello World!</h1>‘;
8       this.innerHTML = ’<h1>Hello World!</h1>’;
9   }
```

Listato 3.2: Esempio di isolamento tramite Shadow DOM di un generico Web Component.

Come si può vedere nell'esempio, lo Shadow DOM puó essere dichiarato in maniera `open` oppure `close`, l'uso di questa seconda modalità consente di nascondere lo *shadowRoot* al *Light DOM* al fine di proteggersi da manipolazioni indesiderate, questo impedisce però anche ad alcune tecnologie di accedere ai contenuti, come per esempio ai crawler. Se non vi sono particolari esigenze è consigliata l'adozione di una modalità "aperta" [14].

Per personalizzare il contenuto di un Web Component a partire dal *Light DOM*, è possibile sfruttare il tag `<slot>`, definito assieme al tag `<template>` dallo standard HTML Template.
All'interno dello *shadow root* possono essere dichiarati un qualsiasi numero di elementi *slot*, questi vengono identificati dal loro attributo *name* e permettono, a chi utilizza il componente, di definire facilmente il contenuto col quale sostituire i placeholder dichiarati in base alle esigenze correnti.

3.6 Architettura finale

A seguito della formulazione rigorosa del processo di trasformazione intrapreso dal gruppo e dopo aver definito gli aspetti principali dell'approccio a micro-frontend che si è deciso di adottare, è possibile procedere con la formalizzazione dell'architettura finale del sistema.
Verranno quindi illustrati nel seguito i principali servizi identificati fino a questo momento, per poi proseguire con lo studio delle tecnologie più adatte a supporto dell'evoluzione del sistema in essere. In conclusione, verrà ri-

portato in maniera riassuntiva il disegno finale del sistema, identificando i principali attori e le relazioni che intercorrono tra di essi.

3.6.1 I servizi presenti

Come affermato in sezione 3.4.2, il processo di decomposizione in servizi dei prodotti avverrà in più passaggi, questo è in parte dovuto alla considerevole dimensione dei sistemi coinvolti. Nonostante ciò, sono già avvenuti numerosi confronti sull'argomento da parte degli addetti ai lavori. Attualmente, le prime sessioni di analisi hanno portato all'individuazione dei componenti descritti nel seguito. Quelli elencati però rappresentano solo una parte degli elementi che andranno a comporre il sistema finale e alcune decisioni potranno anche essere ridiscusse con l'evolversi della situazione.
Tra le funzionalità attualmente sotto analisi che comportano lo sviluppo di un nuovo micro-frontend, vi sono le seguenti:

- *Servizi Demografici*, funzionalità di consultazione, sia attuale che storica, dei dati anagrafici, unita alla possibilità di ottenere autocertificazioni e certificazioni con timbro digitale.
  Presente su J-City Gov e Sportello Telematico.

- *Amministrazione trasparente*, servizio per favorire la partecipazione dei cittadini alle attività della Pubblica Amministrazione attraverso la pubblicazione di piani, informazioni, documenti e atti previsti dal Decreto Legislativo 33/2013.
  Presente su J-City Gov.

- *Pagamento multe*, servizio integrato con Concilia, ossia il software Maggioli per la Polizia Municipale, fornisce un sistema di interfacciamento che permette al cittadino di effettuare il pagamento delle contravvenzioni relative alle violazioni al codice della strada.
  Presente su J-City Gov e Municipium App.

- *Presentazione pratiche*, servizio che consente ai cittadini di compilare online in completa autonomia i moduli digitali, per la presentazione di istanze riguardanti comunicazioni e richieste appartenenti a vari domini differenti, tra questi quelli inerenti ai servizi scolastici, al codice della strada, al S.U.E. (Sportello Unico Edilizia) ed al S.U.A.P. (Sportello Unico Attività Produttive).
  Presente su J-City Gov e Sportello Telematico.

- *Calcolatrice IMU*, funzionalità pensata per consentire ai cittadini di effettuare il calcolo riguardante l'imposta municipale unica (IMU) e successivamente procedere con la stampa del modello F24 relativo oppure direttamente con il pagamento online.
  Presente su J-City Gov.

- *Calcolatrice TARI*, funzionalità pensata per consentire ai cittadini di effettuare il calcolo riguardante la tassa sui rifiuti (TARI) e successivamente procedere con la stampa del modello F24 relativo oppure direttamente con il pagamento online.
  Nuova funzionalità.

- *Sportello Virtuale*, servizio pensato per consentire all'Ente di erogare l'attività di sportello per qualsiasi tipo di tematica, senza imporre al cittadino di presentarsi presso i propri uffici.
  Nuova funzionalità.

In ambito di microservizi è doveroso fare una distinzione tra quelli che saranno in carico al gruppo dei servizi al cittadino e quelli invece che si possono definire esterni al gruppo, i primi attualmente sono:

- *Reportistica*, utile alla generazione di modelli scaricabili dal cittadino, come ricevute e certificati. Utile ai servizi demografici, presentazione pratiche, pagamento multe, calcolo IMU e TARI.

- *Timbro*, pensato per l'apposizione di timbri digitali a qualsiasi genere di documento per cui lo si ritenga necessario. Utile ai servizi demografici e di presentazione pratiche.

- *Viario*, adibito alla persistenza e al recupero, o alla validazione, di informazioni collegate agli stradari cittadini, al fine di suggerire e verificare indirizzi durante la compilazione delle richieste, effettuate dagli utenti, su determinati campi, come indirizzo di nascita o residenza. Utile ai servizi demografici, presentazione pratiche, pagamento multe, calcolo IMU e TARI.

Tra i servizi non in gestione alla divisione servizi al cittadino, non necessariamente basati su una architettura a microservizi, sono elencati, sia quelli in mano a team aziendali, sia quelli non appartenenti al gruppo Maggioli. Si fa notare che gli applicativi elencati in seguito, forniscono tutti servizi con i quali uno o più prodotti del gruppo già possiedono un'integrazione e sono:

- *Sicr@web*, nello specifico:

  - *J-Demos*, sistema informativo per l'Ufficio Servizi Demografici, è il software per la gestione dei servizi di Anagrafe, ANPR, Elettorale, Stato Civile, Leva, Statistica e Carta d'Identità Elettronica (CIE).
    Integrato con i servizi demografici presenti su J-City Gov e Sportello Telematico.
  - *J-Trib*, sistema informativo per i Servizi Tributari e a Domanda Individuale, fornisce quindi pieno supporto alla gestione dei servizi tributari quali Ici/Imu, Tari, Icp, Dpa e Tosap.
    Integrato con i servizi di calcolo tributi presenti su J-City Gov.
  - *J-Iride*, sistema informativo per la gestione e semplificazione dei flussi documentali e procedimenti amministrativi, pensato per la gestione delle informazioni, dei documenti, dei processi e dei procedimenti amministrativi.

Integrato con i servizi di presentazione pratiche di J-City Gov e Sportello Telematico.

  - *J-PE*, sistema informativo per la gestione delle pratiche edilizie, dedicato alla gestione delle pratiche edilizie in conformità alle normative nazionali, regionali, provinciali e ai regolamenti comunali. Integrato con i servizi di presentazione pratiche presenti su J-City Gov e Sportello Telematico.

- *Concilia*, applicativo pensato per la Polizia Municipale e Locale, facilita la gestione delle violazioni al Codice della Strada e delle sanzioni extra CdS.
  Integrato con i servizi di pagamento multe presenti su J-City Gov e Municipium App.

- *ANPR Service*, servizi legati all'Anagrafe Nazionale della Popolazione Residente, banca dati nazionale che offre i servizi demografici per favorire la digitalizzazione e il miglioramento dei servizi a Cittadini.
  Integrato con i servizi demografici presenti su J-City Gov e Sportello Telematico.

- *AuthService*, servizio centralizzato di autenticazione.
  Integrato con tutti e tre i prodotti.

- *J-City Gov PagoPA (JPPA)*, come si evince dal nome in origine era in carico al team di J-City Gov, consiste nel servizio di gestione dei pagamenti.
  Integrato con i servizi di pagamento presenti su J-City Gov e Municipium App.

3.6.2 Lo stack tecnologico di riferimento

Prima di concludere con la definizione del processo di trasformazione illustrando il disegno finale, è necessario presentare le soluzioni tecnologiche

che si intende adottare e le motivazione che hanno condotto a queste scelte, principalmente in funzione dei requisiti e dei vincoli descritti.

Il primo vincolo imposto dal senior management riguarda l'adozione di Spring Boot[11] come framework di sviluppo dei servizi di backend, ovviamente BFF inclusi.
Questa scelta deriva dal fatto che Java è il linguaggio di programmazione di riferimento storico per l'azienda. Di fatto la maggior parte dei prodotti sono basati su questa tecnologia, il che ha portato gran parte degli sviluppatori dell'azienda a maturare competenze ed esperienza sul linguaggio, queste rappresentano dunque per l'azienda un patrimonio importante di cui valersi e da coltivare.
Partendo dalla scelta di questo linguaggio, sono state compiute, in un periodo più recente, le indagini e le analisi per individuare le tecnologie più adatte per lo sviluppo dei microservizi aziendali all'interno dei progetti futuri. Al termine di queste, in base ai dati raccolti, i vertici aziendali hanno deciso di adottare appunto Spring Boot come framework di riferimento.

In materia di framework JavaScript lato client per la realizzazione di interfacce utente dedicate ad applicazioni web, la scelta aziendale è quella di convergere su Angular.
Le radici di questa decisione sono questa volta più recenti e derivano dalle valutazioni svolte durante le attività di aggiornamento di Sicr@web, come descritto in sezione 2.3, e la sua migrazione verso tecnologie web per lo sviluppo della relativa UI.

Come cloud provider è vincolante adottare Google Cloud e i prodotti offerti all'interno della relativa suite di servizi di cloud computing, ossia Goolgle Cloud Platform[12] (GCP).
L'azienda ha infatti deciso di affidarsi a Google e ai suoi strumenti per gestire questo genere di esigenze e non solo. Per questo motivo sono stati proposti

[11] `https://spring.io/projects/spring-boot`
[12] `https://cloud.google.com/`

vari corsi di formazione ai dipendenti, in particolare su Google Kubernetes Engine e tutto l'ecosistema collegato ad esso.

Il gruppo dei servizi al cittadino ha valutato attentamente le direttive imposte dall'azienda e ritenuto logiche e coerenti le tecnologie proposte, ad eccezione della decisione di adottare Angular come unica soluzione lato frontend.
Il framework è utilizzato su Municipium ma meno conosciuto dai restanti team, infatti il gruppo di Sportello Telematico non ha esperienza in materia, mentre quello di J-City Gov possiede solamente conoscenze limitate della precedente versione AngularJS, in quanto adottato in contesti circoscritti. Inoltre, valutare altre soluzioni potrebbe rivelarsi utile, visto anche che, come illustrato in sezione 3.4.1, tra gli strumenti di Designers Italia a disposizione degli sviluppatori esiste il progetto React Kit, basato appunto su React, che potrebbe essere utile impiegare all'interno dei micro-frontend, a differenza dell'alternativo progetto Angular che pare essere attualmente non più manutenuto.
Queste le ragioni principali che hanno portato il gruppo a chiedersi se non fosse una scelta prematura legarsi a questa tecnologia.

I dubbi nati in seno al gruppo hanno condotto alla decisione di effettuare delle più attente valutazioni in materia. Per questa ragione, sono stati compiuti ulteriori studi ed un'analisi approfondita anche su framework alternativi, principalmente React e Vue.js[13].
Gli approfondimenti condotti hanno portato alla scelta di abbracciare la decisione aziendale. Infatti, questa soluzione fornisce il vantaggio di agevolare i vari team di dominio nel prendersi in carico i micro-frontend, in quanto sviluppati con tecnologie loro note. Inoltre, la decisione di adottare una composizione basata su Web Component si sposta bene con il framework, dato che fornisce la possibilità di creare componenti direttamente sotto forma di Custom Element, chiamati Angular Element.

[13] `https://vuejs.org/`

Angular Element[14] è una funzionalità disponibile tramite il pacchetto *@angular/elements*. Questo pacchetto espone il metodo `createCustomElement` che può essere appunto utilizzato per convertire un componente Angular qualsiasi in un Custom Element.
Tutto ciò rende facile ed immediato generare nuovi Web Component, inoltre consente di collegare automaticamente le potenzialità offerte da Angular con le equivalenti funzionalità HTML standard, creando un ponte tra le due tecnologie di componenti e i loro cicli di vita.

3.6.3 Disegno finale

A seguito delle valutazioni effettuate e delle considerazioni espresse in precedenza, è possibile procedere illustrando quello che è il disegno finale del sistema, evidenziando le principali componenti identificate e le relazioni che le legano.

In figura 3.4, viene riportata la struttura che dovrà assumere il sistema, a partire dall'attuazione della seconda parte della prima fase del processo di trasformazione identificato.

In particolare, viene riportato il contesto di utilizzo dei micro-frontend, realizzati come Angular Element, che andranno a contenere le funzionalità di dominio precedentemente sviluppate all'interno dei singoli applicativi. Questi, in una prima fase avranno il compito di includere i componenti realizzati sino a quel momento e di continuare a ricoprire il proprio ruolo di orchestratori. Ruolo che in seguito, con la seconda fase del processo di trasformazione, verrà interpretato dall'applicativo definitivo.

Sempre in figura, vengono poi mostrati, il BFF incaricato di fornire una astrazione delle API al relativo micro-frontend, basato sul framework Spring Boot, e i servizi di back-office relativi allo specifico dominio del componente di frontend in questione. In questo caso viene mostrato Sicr@web in rappre-

[14] `https://angular.io/guide/elements`

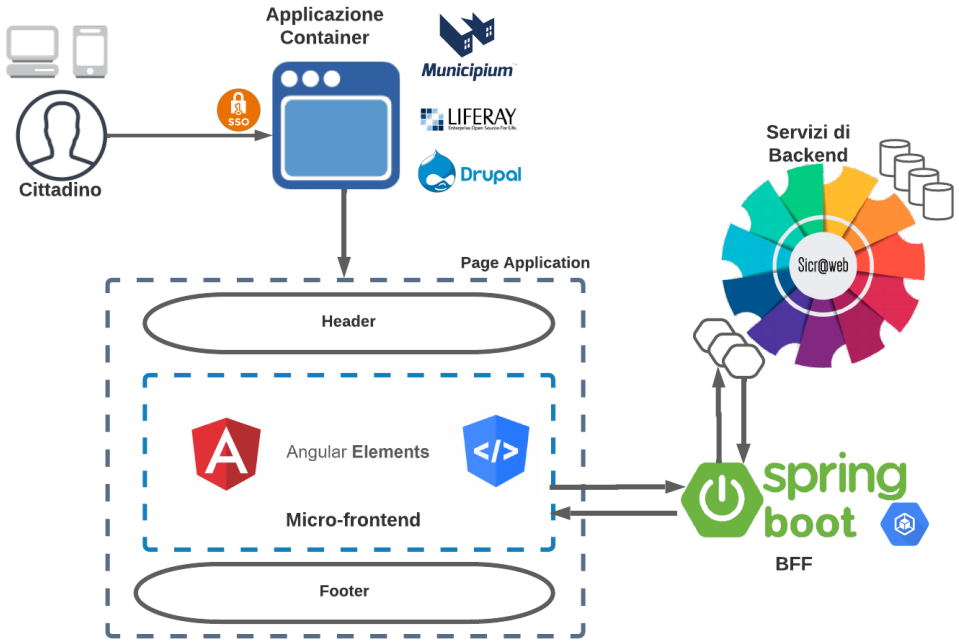

Figura 3.4: Riepilogo del disegno finale della struttura del sistema.

sentanza dei vari possibili interlocutori, come Concilia o JPPA, elencati in sezione 3.6.1.

Capitolo 4

Definizione del processo di automazione

Tempo e risorse sono vincoli critici all'interno di qualsiasi contesto aziendale, per questo motivo è fondamentale che le aziende siano in grado di reagire in maniera tempestiva alle esigenze di mercato e con il livello di qualità che la crescente base di utenti si aspetta. Ormai nessuna organizzazione può permettersi di convivere con un ciclo di rilascio del software caratterizzato da attività manuali, ripetitive e soggette ad errori [37].
In sezione 1.2 è già stato annunciato quanto sia implorante adottare una cultura dell'automazione con questo genere di architetture, visto anche il maggior numero di artefatti indipendenti con cui ci si ritrova a che fare in questi contesti [12].

Una discussione completa ed approfondita sull'argomento DevOps esula dallo scopo di questo testo, tuttavia all'interno del capitolo verranno analizzati alcuni aspetti importanti correlati al progetto e alla tipologia di architettura adottata. Il lavoro presentato non deve però essere considerato come il riassunto di quella che è la gestione definitiva del processo di automazione legato al sistema, piuttosto descrive quelle che sono le considerazioni e le decisioni prese in merito a questo argomento all'interno di un processo tut-

tora in divenire, corredato da importanti nozioni generali in materia ed altre considerazioni strettamente collegate alle architetture a micro-frontend.

Per queste ragioni all'interno del capitolo verranno presentati per cominciare i concetti, le politiche e le direttive aziendali che hanno portato a definire le modalità di organizzazione del lavoro del gruppo Servizi al cittadino, successivamente verranno discusse le principali fasi del processo di integrazione prima e di distribuzione continua poi, per terminare con una visione d'insieme dell'organizzazione generale della gestione della pipeline di CI/CD pensata per i componenti del caso di studio.

4.1 Organizzazione del lavoro

Al fine di ottenere i massimi vantaggi dal processo di automazione, è importante definire le regole di gestione del lavoro in maniera chiara e rigorosa. In questa sezione verrà dunque descritta la modalità di organizzazione del lavoro, discutendo quelli che sono i confini di libertà in mano agli sviluppatori dei team e quelli che invece sono i vincoli imposti dal management aziendale. Verranno presentate le politiche interne del gruppo Servizi al cittadino e gli strumenti aziendali messi a disposizione dei team.

4.1.1 Organizzazione aziendale

Specialmente in passato, molte azienda hanno organizzato la propria struttura interna affidando la gestione delle strategie di automazione ad una cerchia ristretta di figure appartenenti all'organizzazione, detentori della conoscenza dell'intero funzionamento del ciclo di rilascio, consentendo ad ancora meno individui la possibilità di modificare l'infrastruttura adibita alla generazione e alla distruzione degli artefatti.
Di conseguenza, gli sviluppatori hanno trattato le proprie pipeline di automazione come una scatola nera, impossibile da modificare ma indispensabile al fine del rilascio del software.

Questo genere di approccio è stato ampiamente criticato e recentemente, grazie alla sempre maggior diffusione della cultura DevOps, queste situazioni stanno diventando meno frequenti [34].
Nel contesto di architetture a micro-frontend è importante concedere ai team di sviluppo la possibilità di modificare ed adattare la pipeline su cui si sta operando e non limitare il loro lavoro esclusivamente alla scrittura del codice [12]. In generale, in un contesto di questo tipo non è possibile fare affidamento esclusivamente sulla medesima modalità di gestione, particolarmente in fase di compilazione, questo vista anche la possibilità di adottare uno stack tecnologico differente per ogni componente.
Centralizzare tutte le decisioni ed imporre queste in tutte le situazioni, potrebbe portare ad un risultato finale peggiore, rispetto ad una modalità di lavoro basata sul fornire il giusto grado di libertà di azione agli sviluppatori.

Un valido approccio consiste nell'istituire una solida strategia di automazione attraverso la definizione di un'insieme di vincoli validi per tutta l'organizzazione, al fine di indirizzare i singoli team verso la giusta direzione. I vincoli definiscono i confini entro il quale i singoli team possono operare, dovrebbero essere individuati da chi detiene la leadership tecnologica, in collaborazione con architetti ed ingegneri infrastrutturali. La loro introduzione non significa ridurre la libertà degli sviluppatori, serviranno invece per guidarli verso l'utilizzo degli standard dell'azienda, astraendoli il più possibile dal loro mondo, consentendo loro di innovare all'interno di questi confini [12]. É fondamentale trovare il giusto equilibrio durante la definizione di queste direttive ed assicurarsi che tutti capiscano non soltanto come applicarle ma anche il perché si sono rese necessarie.

L'approccio adottato da Maggioli coincide con quanto descritto, infatti all'interno dell'azienda un insieme di figure esperte, appartenenti al team aziendale di infrastruttura e al gruppo di Ricerca e Sviluppo, ha definito quelli che sono gli standard e le linee guida da adottare, garantendo però la possibilità di introdurre le opportune modifiche ed ottimizzazioni.

Gli strumenti e i tool principali di CI/CD, che verranno presentati nelle sezioni seguenti, sono quindi loro responsabilità, ma tutti gli script e i passaggi per generare un artefatto sono di proprietà del singolo team di sviluppo, questo perché, soltanto i membri di quest'ultimo, possono conoscere il modo migliore per arrivare a produrre un artefatto il più possibile ottimizzato con il codice scritto.

Esattamente come le altre parti della strategia di automazione, le direttive imposte non dovrebbero essere statiche, ma devono essere riviste, migliorate ed aggiornate di pari passo con l'evoluzione dell'azienda e del mondo circostante. Le figure preposte in azienda a questo compito sono quelle incaricate di promuovere questi cambiamenti, oltre che di verificarne la loro adozione.
Non meno importante il fatto di incoraggiare una cultura di condivisione e innovazione, creando momenti in cui i team possono scambiarsi idee e soluzioni, ciò si rende particolarmente importante quando si lavora in un ambiente distribuito.

4.1.2 Organizzazione del team

All'interno della divisione dei servizi al cittadino, la gestione e l'organizzazione delle attività è compito dei singoli referenti di ciascuno dei tre team, in collaborazione con il responsabile del gruppo.
I singoli team sono organizzati in modo che ogni sviluppatore sia in grado di operare in maniera trasversale all'interno dei progetti, dallo sviluppo alla gestione della pipeline, ciononostante in fase di assegnazione delle attività si tiene conto dell'esperienza e delle conoscenze individuali.

Come più volte discusso la condivisione delle informazioni e delle conoscenze tra i lavoratori sono fortemente incentivate dall'azienda, per questo motivo gli incontri tra i membri del team sono promossi e ben organizzati, al fine di garantire la massima efficienza ed utilità.
Sono quindi previsti stand-up meeting giornalieri di una durata approssimativa di quindici minuti e degli incontri settimanali di circa un'ora, con il fine

di condividere lo stato di avanzamento dei lavori. Durante questi incontri non vengono affrontati in maniera esaustiva i problemi riscontrati, i quali sono invece discussi in sessioni apposite di lunghezza variabile, organizzate all'occorrenza, solo per i membri del team coinvolti direttamente.
Sono inoltre previsti degli incontri formali al raggiungimento di ogni milestone, organizzati anche tra sedi diverse in base alle esigenze, con l'obiettivo principale di condividere lo stato attuale del progetto e di effettuare delle valutazioni critiche sui procedimenti adottati, al fine di individuare azioni correttive.

4.1.3 Strumento di gestione del lavoro

Allo scopo di organizzare adeguatamente le attività si rendere utile utilizzare uno strumento in grado di tenere traccia e mantenere uno storico delle richieste, delle anomalie individuate e più in generale dei progressi raggiunti e degli incarichi da svolgere, tutto ciò a supporto della pianificazione e programmazione del lavoro da compiere.
Lo strumento adottato per questo progetto è Jira[1] in quanto standard aziendale. Jira fa parte di una famiglia di prodotti studiati per aiutare qualsiasi tipo di team a gestire il lavoro, in origine era dedicato esclusivamente al monitoraggio di bug e ticket ma oggi si è evoluto in un potente strumento di gestione del lavoro per tutti i casi di utilizzo, fornisce diversi tipi di report, aiuta ad analizzare i progressi, i problemi, gli ostacoli e le risorse di qualsiasi progetto.

Per registrare i problemi emersi durante il progetto non ancora risolti e mantenere uno storico di quelli già affrontati viene utilizzato il concetto di *issue*. Queste possono essere registrare in maniera agevole all'interno dello strumento, inoltre è possibile specificare vari parametri in modo da descrivere tutti gli aspetti fondamentali. Tra i parametri principali troviamo un titolo, una descrizione, la tipologia (MEV, MAD, MAC), il reporter e l'assegnatario,

[1] `https://www.atlassian.com/software/jira`

la data di creazione, il tempo stimato e quello registrato, inoltre è possibile definire una priorità ed indicare la versione di correzione, aggiungere un'analisi RID e cambiarne lo stato secondo il flusso di lavoro definito.
Le issue possono essere commentate e ogni utente interessato può registrarsi come osservatore per ricevere una notifica, in modo da rimanere aggiornato sui cambiamenti che avvengono, inoltre è anche possibile collegare tra loro le issue secondo varie relazioni in base alle esigenze.

Jira è fortemente personalizzabile per rispondere a tutte le esigenze, inoltre uno dei suoi vantaggi è quello di fornire automaticamente report standard senza alcuna configurazione necessaria. Questi report standard comprendono un'ampia gamma di diagrammi di reporting come il monitoraggio del tempo, il carico di lavoro e anche report astratti come i grafici a torta che possono essere utilizzati in vari modi.

4.1.4 Ambienti

All'interno delle organizzazioni di medie e grandi dimensioni, la strategia più comunemente adottata consiste nel predisporre una combinazione di ambienti di test, staging e produzione [12].
Con il termine ambiente si fa comunemente riferimento al contesto di esecuzione del software, il quale racchiude un installazione completa ed isolata del sistema sviluppato. A seconda dello scopo per cui sono stati predisposti, i vari ambienti possono possedere delle caratteristiche differenti.

Gli ambienti descritti nel seguito, accompagnati da una descrizione del loro scopo, sono quelli attualmente inclusi nel processo di automazione del progetto:

- *Locale*, rappresenta l'ambiente utilizzato durante lo sviluppo del codice da ogni singolo sviluppatore e consiste nella macchina personale in dotazione a ciascuno di essi. La configurazione di questi ambienti comporta

l'individuazione del miglior compromesso tra fedeltà, rispetto all'ambiente finale di produzione, e complessità legata alla manutenzione e prestazioni fornite a supporto del lavoratore.

- *Staging*, o pre-produzione, ambiente su cui effettuare il collaudo delle nuove funzionalità prima che queste vengano effettivamente dispiegate in produzione, per questo motivo dovrebbe assomigliare il più possibile a quest'ultimo.

- *Produzione*, ambiente finale in cui gli utenti possono collegarsi per accedere ai servizi del sistema. Rappresenta il punto definitivo raggiunto al termine del processo di rilascio e deve ospitare soltanto le versioni dell'applicativo che hanno superato tutti i rigorosi test e controlli predisposti.

Nelle prossime sezioni verrà illustrato il processo di automazione ideato, analizzando la relazione tra gli ambienti e le varie fasi del ciclo di rilascio.

4.2 Controllo di versione

Durante la fase di progettazione del processo di automazione, è fondamentale analizzare e decidere quali tecniche e quali strumenti adottare in merito al sistema di controllo di versione o VCS (Version Control System) e alla sua gestione. I sistemi di controllo di versione più adottati sono Git, Mercurial, Subversion (SVN) e Perforce.
In azienda è stato a lungo adottato SVN come strumento di riferimento, di recente la scelta aziendale è stata quella di migrare su Git, attualmente questa soluzione rappresenta un vincolo tecnologico ed in particolare viene imposto anche l'uso della piattaforma GitLab[2] privata Maggioli.
Collegate alla scelta di questo strumento esistono varie opzioni riguardo alle politiche di gestione che possono essere adottate, all'interno di questa sezione

[2] `https://about.gitlab.com/`

verranno valutate alcune possibili alternative ed indicate le scelte attualmente prese.

4.2.1 Organizzazione del repository

In fase di definizione del sistema di controllo di versione relativo al proprio progetto è utile decidere anche quale approccio adottare in termini di organizzazione del repository. In questo contesto, la scelta che occorre compiere è se utilizzare una strategia basata su una gestione a monorepo oppure su una gestione a polirepo.
All'interno di una architettura a micro-frontend sono percorribili entrambe le strade, la soluzione migliore consiste nel far ricadere la propria scelta sul modello di organizzazione che offre maggiori vantaggi in funzione del contesto specifico.

Un approccio a *monorepo* si basa sul fatto che tutti i team utilizzano lo stesso repository di versionamento e quindi tutti i progetti vengono ospitati assieme. Uno dei principali punti di forza di questo modello consiste nella sua capacità di condivisione del codice, è fondamentale però impiegare un impegno costante nel garantire un elevato livello di qualità nell'organizzazione del codice ed evitare di infrangere il principio di indipendenza tra i micro-frontend, mantenendoli adeguatamente disaccoppiati. In questo senso esistono vari strumenti che possono corre in aiuto, tra questi vi sono Lerna[3] e il progetto Nx[4] di Nrwl.
Lerna ottimizza il flusso di lavoro relativo alla gestione di codebase di grandi dimensioni, organizzate in pacchetti separati versionati in maniera indipendente, consente infatti di gestire tutte le dipendenze comuni e di pubblicare un pacchetto quando una nuova versione è pronta per essere rilasciata [12].
Nx (Nrwl Extensions) è un altro strumento molto utile in questo contesto che aiuta ad adottare una gestione strutturata del codice condiviso, consente di

[3] `https://lerna.js.org/`
[4] `https://nx.dev/`

impostare delle restrizioni di accesso ed aiuta ad ottimizzare i tempi in fase di sviluppo, compilando solo le parti interessate dalla modifiche apportate ed evitando quindi di effettuare operazioni non necessarie [38].

L'opposto di una strategia monorepo è il *polyrepo*, o multi-repository, in cui ogni applicazione viene gestita in maniera isolata all'interno del proprio specifico repository.
Questa strategia può rivelarsi particolarmente adatta ad un approccio basato su micro-frontend, il rischio però è di incorrere in una eccessiva proliferazione di repository, in particolare per i progetti caratterizzati da una divisione orizzontale, in cui l'applicazione può risultare composta da un elevato numero di parti diverse [12].

In generale ogni scelta possiede i propri vantaggi e i propri svantaggi [39]. Nel caso di una gestione a monorepo, vi è un significativo vantaggio in termini di visibilità: trovare esempi, aggiornare dipendenze e scoprire servizi utili già esistenti risulta più semplice con questo approccio. Al contrario, i sistemi multi-repository consentono un'elevata flessibilità e stabilità oltre che una maggiore indipendenza.

All'interno del contesto aziendale vengono principalmente adottate soluzioni basate su approcci a monorepo, molte di queste fanno affidamento su Nx, strumento molto apprezzato da vari team appartenenti all'organizzazione.
Per il progetto corrente però, la scelta è ricaduta sul ricorrere ad una gestione a multi-repository. Questo essenzialmente è dovuto alla forte esigenza di indipendenza ed autonomia, alle caratteristiche del progetto, che necessita di integrare sistemi legacy diversi, e ai problemi di carattere logistico ed organizzativo, conseguenti dall'eventuale utilizzo di un approccio a monorepo all'interno del contesto corrente, caratterizzato dalla presenza di un elevato numero di team differenti dislocati sul territorio.

4.2.2 Modello di branching

Un modello di branching definisce le strategie e le politiche legate alla creazione e alla gestione dei rami di sviluppo di Git, determinando le modalità secondo cui gli sviluppatori devono operare. Il modello suggerito in azienda è GitFlow, presentato da Vincent Driessen nel 2010 [40].

L'approccio dettato da GitFlow si basa su un insieme di concetti, in primo luogo il repository centrale deve contenere due rami principali, il cui ciclo di vita segue quello di tutto il progetto registrandone l'avanzamento. Il primo è il ramo *master* dove il codice sorgente contenuto riflette sempre lo stato pronto per la produzione, il secondo è *develop*, definito anche "integration branch", dove vengono integrate le funzionalità in sviluppo. Quando il codice presente nel ramo *develop* raggiunge un punto stabile ed è pronto per essere rilasciato, tutte le modifiche vengono portate su *master* e contrassegnate da un numero di versione.

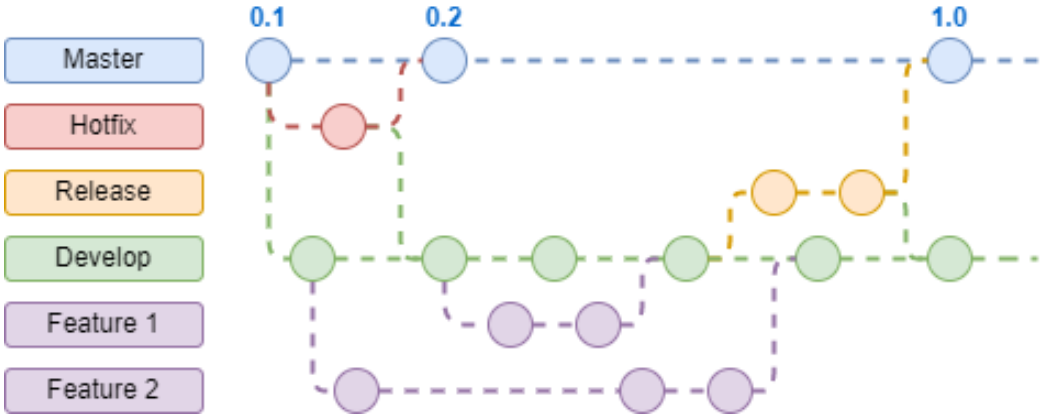

Figura 4.1: Modello di branching adottato.

Accanto ai rami principali appena descritti, durante il ciclo di sviluppo ne vengono utilizzati altri di supporto per permettere ai membri del team di eseguire attività in parallelo. A differenza dei rami principali, questi hanno

sempre una vita limitata, poiché prima o poi verranno rimossi.
I diversi tipi di rami che possono essere utilizzati sono:

- *feature branches*: vengono utilizzati per sviluppare nuove funzionalità per le prossime versioni. Questi rami nascono da *develop* e al termine vengono nuovamente fusi in esso.
- *release branches*: servono a isolare la fase di rilascio e consentono agli altri membri del team di continuare a lavorare senza rischiare di includere modifiche non volute. Nascono da *develop* quando sono state sviluppate le funzionalità necessarie, al termine della preparazione le modifiche vengono riportate in *master* e in *develop*.
- *hotfix branches*: quando viene rilevata un'anomalia critica in produzione che necessita di essere risolta immediatamente, si può intervenire creando questo tipo di ramo direttamente da *master* e dallo stato della versione in questione. Una volta che è stata effettuata la correzione le modifiche vengono riportate sia in *master* e che in *develop*.

4.3 Continuous Integration

La Continuous Integration (CI), è una pratica di sviluppo software basata sul promuovere l'integrazione del lavoro svolto dai membri del team in maniera frequente, evitando quindi di mantenere a lungo le modifiche apportate esclusivamente all'interno dell'area di lavoro locale [37].
Ogni integrazione viene verificata da un processo automatizzato al fine di rilevare l'insorgere di errori il più rapidamente possibile, il tutto in maniera continua e ripetuta durante tutto il ciclo di sviluppo.
Molti ritengono che questo approccio porti a problemi di integrazione significativamente ridotti e consenta a un team di sviluppo di collaborare in maniera più efficiente ed efficace [41].

All'interno di una architettura a micro-frontend, in materia di CI, è possibile adottare strumenti differenti, come detto in sezione 4.1.1, la cosa im-

portante è fornire la dovuta libertà di azione ai singoli team all'interno dei confini definiti.
In questo contesto, introdurre nuove tecnologie non deve rappresentare un problema, ovviamente non è consigliato adottare una moltitudine di strumenti differenti, infatti è possibile promuovere vari di questi da utilizzare come standard all'interno di tutti i progetti ma questo non deve tuttavia rappresentare un ostacolo al processo di innovazione.

All'interno di questa sezione verranno quindi presentate le più comuni attività di controllo che il processo automatizzato deve compiere, al fine di garantire il successo dato dall'utilizzo di questa pratica.

4.3.1 Compilazione

L'obiettivo di questa fase è di accertarsi che il codice sorgente caricato sul repository continui ad essere correttamente compilato a fronte delle modifiche apportate. Vi sono numerosi strumenti in grado di svolgere questo compito, pertanto a seconda del linguaggio di riferimento esistono spesso più alternative di cui è possibile servirsi.
Lo scopo principale rimane sempre quello di effettuare la compilazione del codice sorgente al fine di generare l'artefatto finale, tendenzialmente però questo genere di strumenti non si limitano esclusivamente a questa funzione ma si occupano anche di reperire automaticamente le risorse necessarie a completare l'esecuzione del programma, della generazione della documentazione e dei report e di verificare i casi di test correlati, assumendo quindi un ruolo chiave all'interno del processo di integrazione continua.
Come anticipato, nella scelta di questi strumenti non sono stati imposti vincoli assoluti, nel seguito sono descritti quelli più utilizzati in azienda a seconda del contesto, adottati come riferimento anche per lo sviluppo dei componenti del progetto.

Per quanto riguarda la gestione dei BFF, scritti dunque in linguaggio Ja-

va, le tecnologie più diffuse sono Gradle[5] e i progetti Apache Ant[6] e Maven[7], è proprio quest'ultimo lo strumento più diffuso in azienda adottato all'interno dei progetti Java.
Apache Maven adotta il paradigma *convention over configuration*, ossia una politica ispirata al principio *Don't repeat yourself* (DRY) che richiede allo sviluppatore di specificare solo gli aspetti che si differenziano dalle implementazioni standard, è quindi basato su un componente di configurazione fondamentale ossia il Project Object Model (POM), un file *XML* contenente la dichiarazione della struttura del progetto e l'insieme delle dipendenze.

Gestione differente per i micro-frontend, dove deve essere eseguito il *transpiling* del codice Typescript, questa operazione avrà il duplice compito di controllo degli errori e di generazione del relativo codice JavaScript, il quale potrà essere correttamente interpretato all'interno del browser. Questo e non solo i compiti affidati alla Angular CLI che tra i vari strumenti si serve di Webpack[8], per raggiungere il proprio obiettivo e assolvere correttamente al suo compito di impacchettamento dei moduli web.
L'attuale versione del framework, non offre però funzionalità specifiche legate al contesto appositamente pensate per Angular Elements, dove la gestione dei risultati del processo di compilazione può essere automatizzata ulteriormente, si consiglia infatti di raggruppare le risorse JavaScript generate, concatenandole in unico file, in modo che sia più facile includere il codice necessario per rendere disponibile il Web Component all'interno della pagina.
Questa operazione può essere compiuta in modi diversi, una strada possibile è quella di sfruttare il progetto *ngx-build-plus*[9], il quale estende il comportamento predefinito adottato dalla Angular CLI, consentendo di genere un unico pacchetto finale. Un approccio differente viene invece mostrato nel listato 4.1, realizzato attraverso uno script personalizzato contenuto all'in-

[5] `https://gradle.org/`
[6] `https://ant.apache.org/`
[7] `https://maven.apache.org/`
[8] `https://webpack.js.org/`
[9] `https://github.com/manfredsteyer/ngx-build-plus`

terno di un apposito file di progetto.

```
const fs = require("fs-extra");
const concat = require("concat");

(async function build() {
  const files = [
    "./dist/<PROJECT_NAME>/runtime.js",
    "./dist/<PROJECT_NAME>/polyfills.js",
    "./dist/<PROJECT_NAME>/main.js",
  ];
  await fs.ensureDir("elements");
  await concat(
    files,
    "elements/<PROJECT_NAME>/<PROJECT_NAME>.js"
  );
  await fs.copyFile(
    "./dist/<PROJECT_NAME>/styles.css",
    "elements/<PROJECT_NAME>/styles.css"
  );
  await fs.copy(
    "./dist/<PROJECT_NAME>/assets/",
    "elements/<PROJECT_NAME>/assets/"
  );
})();
```

Listato 4.1: Script personalizzato realizzato per gestire i risultati del processo di compilazione dei componenti Angular.

Infine, è invece disaccoppiata la gestione delle dipendenze, dove è necessario fare affidamento ad un *package manager*, vari quelli a disposizione, tra i più diffusi vi sono Npm[10] e Yarn[11].

[10] https://www.npmjs.com/
[11] https://yarnpkg.com/

4.3.2 Test

Lo scopo di questa fase è di rilevare bug ed eventuali regressioni prima che questi possano raggiungere gli ambienti di produzione. Eseguire le dovute verifiche con regolarità è dunque una pratica essenziale, pertanto è importante introdurre all'interno del processo di integrazione continua passaggi adibiti all'esecuzione dei test, al fine di automatizzare i controlli e quindi di non rallentare eccessivamente la fase di sviluppo.

In questo contesto un approccio a micro-frontend non introduce nuove specifiche esigenze di gestione, non sono quindi richiesti particolari accorgimenti rispetto a quanto già non venga fatto normalmente in materia di test, relativamente alla parte frontend.
Per quanto riguarda i classici test unitari e di integrazione, propri di questa fase della pipeline, una differenza che può presentarsi consiste nel fatto che l'area da testare per team è più limitata in confronto ad una normale applicazione, la relativa complessità potrebbe quindi rivelarsi inferiore. Discorso diverso per i test *End to End*, conosciuti come E2E, propri delle fasi post-rilascio, dove la complessità di gestione può aumentare considerevolmente, specie nel caso si stia adottando una suddivisione orizzontale [12].

La gestione dei test non si limita però alla semplice esecuzione degli stessi ma riguarda anche la generazione della relativa reportistica oltre che la verifica della copertura del codice, chiamata *code coverage*, attività strettamente correlate alla fase di analisi statica.
Esistono molti strumenti diversi che possono essere adottati a supporto di questa fase, in materia di copertura dei sorgenti JaCoCo, per la parte BFF, e Karma, in Angular, sono le due principali tecnologie utilizzate dal gruppo nello specifico frangente.

4.3.3 Analisi statica

L'analisi statica è un processo che consiste nella revisione di codice sorgente ed artefatti, analizzando staticamente struttura e contenuti, quindi

senza la necessità di mettere effettivamente in esecuzione il sistema. Il tutto viene effettuato al fine di rilevare eventuali difetti, in modo da garantire che quanto prodotto sia conforme con gli standard attesi.
L'analisi statica è generalmente efficace per individuare varie tipologie di problemi come:

- Errori di programmazione.
- Vulnerabilità di sicurezza.
- Mancato adempimento a regole e convenzioni imposte.
- Insufficiente affidabilità, dovuta alla mancata conformità con le buone pratiche architetturali e di scrittura del codice.
- Scarsa manutenibilità, spesso legata a concetti come comprensibilità e modularità.

In particolare, all'interno di una architettura a micro-frontend, può risultare molto vantaggioso inserire, durante queste ispezioni, dei controlli legati al contesto, un esempio è la verifica delle dimensioni finali del pacchetto generato e ancora può essere utile esaminare le specifiche versioni adottate per alcune determinate dipendenze comuni.

In generale esistono vari strumenti automatizzati in grado di assistere programmatori e sviluppatori nell'esecuzione di questa attività, in azienda la direttiva ufficiale è quella di adottare SonarQube[12].
SonarQube è una piattaforma open-source per la gestione della qualità del codice sviluppata da SonarSource[13], offre numerose funzionalità a partire della sua versione community e può essere esteso con plugin gratuiti e commerciali. SonarQube è scritto in Java ed è nato inizialmente per operare su progetti scritti con questo linguaggio, attualmente però non si limita più ad esso ed è in grado di supportare vari linguaggi di programmazione tra cui JavaScript e TypeScript [42].

[12] `http://sonarqube.org/`
[13] `https://www.sonarsource.com/`

Ad ogni scansione SonarQube effettuata un'analisi completa del codice collezionando tutti i dati necessari, le informazioni raccolte possono poi essere visualizzate all'interno delle sezioni di riepilogo, la cronologia dei risultati registrati permette inoltre di fornire grafici contenti l'evoluzione dello specifico progetto, un esempio viene mostrato in figura 4.2.

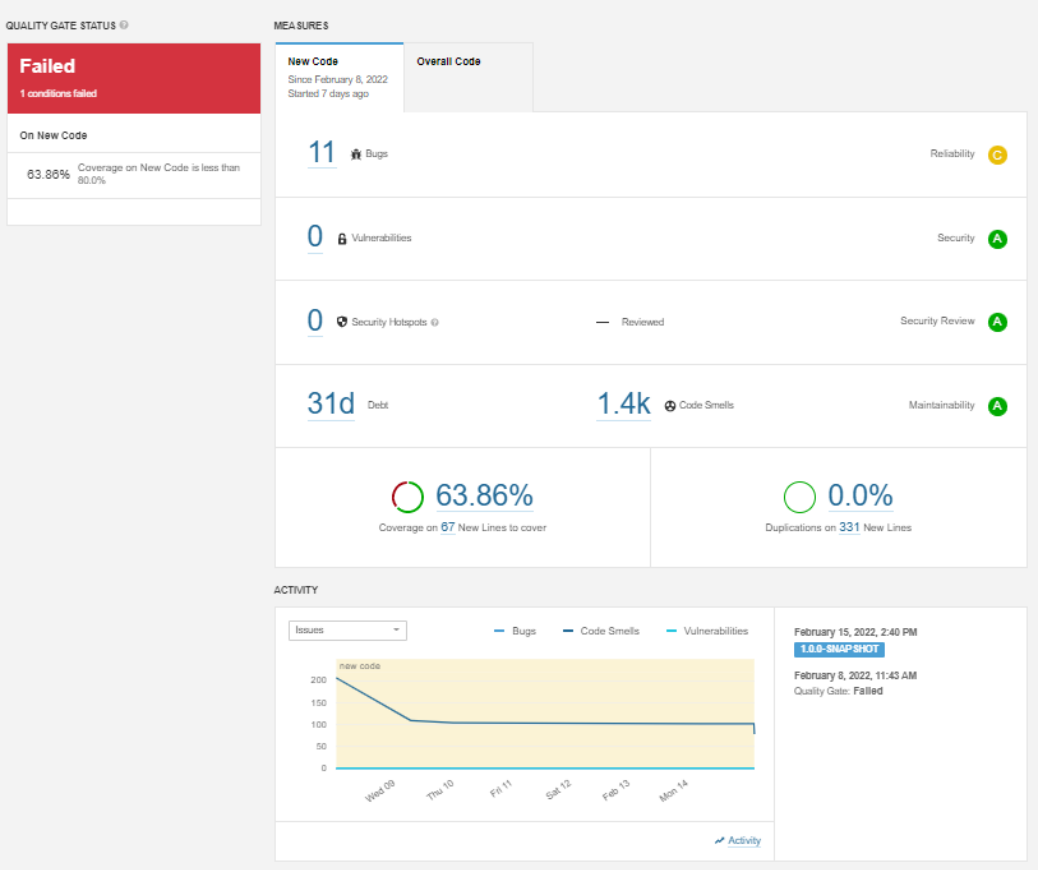

Figura 4.2: Report di analisi presentato da SonarQube all'interno della sezione dedicata allo specifico progetto.

Al termine di questa fase di scansione vengono quindi effettuate tutte le verifiche per assicurare che il codice implementato rispetti gli standard aziendali. Al fine quindi di valutare le nuove modifiche introdotte e sapere se il progetto è pronto per raggiungere gli ambienti di produzione, viene introdot-

to dallo strumento il concetto di *quality gate*, ossia un insieme di condizioni booleane associate alla valutazione di ciascuna metrica, se lo stato finale risulterà *superato* il processo di rilascio potrà proseguire con le fasi successive, altrimenti bisognerà intervenire per sanare la situazione in relazione ai problemi riscontrati.
In tabella 4.1, vengono riportate le soglie associate a ciascuna metrica, configurate in maniera predefinita per i progetti aziendali, le quali possono eventualmente essere personalizzate per ciascuno di essi in base alle esigenze specifiche.

Metrica	**Condizione**	**Valore**
Copertura	inferiore a	70%
Righe duplicate (%)	maggiore di	5,0%
Grado di manutenibilità	peggiore di	A
Grado di affidabilità	peggiore di	A
Hotspot di sicurezza esaminati	inferiore al	100%
Grado di sicurezza	peggiore di	A

Tabella 4.1: Soglie associate a ciascuna metrica.

All'interno dei vari progetti, quindi sia per i micro-frontend sia per i BFF ad essi associati, verranno aggiunti i relativi plugin a seconda del linguaggio corrente, questo con lo scopo di consentire l'integrazione con lo strumento di automazione dello sviluppo e quindi di eseguire l'analisi statica con SonarQube come desiderato.

4.4 Continuous Deployment

Continuous Deployment (CD) è una delle pratiche chiave associate alla filosofia DevOps, prevede una distribuzione automatica e continua del software

verso gli ambienti di produzione e permette di ottenere numerosi vantaggi, come un minor rischio in fase di rilascio, costi inferiori ed una riduzione del *time to market* (TTM).
Questa pratica viene spesso accostata alla Continuous Delivery (CDE), è importante però notare che CD implica l'applicazione di CDE ma che non è vero il contrario, infatti la CDE prevede che il rilascio finale verso gli ambienti di produzione richieda un passaggio manuale di approvazione, cosa non prevista dall'adozione di CD, dove le modifiche apportate vengono propagate direttamente all'ambiente finale senza intervento umano [43].
Nel seguito verranno presentate le soluzioni adottate collegate a questa pratica nonché le tecnologie scelte all'interno del progetto, in funzione delle direttive aziendali in merito.

4.4.1 Pubblicazione

La fase di pubblicazione è un passaggio fondamentale che consiste nell'archiviare tutti gli artefatti generati all'interno di un singolo repository, in modo che questo rappresenti un'unica fonte di verità dal quale attingere per recuperare le risorse ricercate.
Esistono numerosi strumenti che possono essere dispiegati per ospitare gli artefatti generati. A questo scopo a livello aziendale è presente un'installazione di Nexus Repository[14], adibita ad ospitare questo genere di contenuti, in quanto in grado di supportare numerosi formati differenti e quindi di integrarsi con vari *package manager* e strumenti come maven o npm, pertanto sarà la destinazione degli archivi *.jar* e *.war* oltre che dei pacchetti web generati.

Al fine di essere resi accessibili pubblicamente, e quindi integrati nelle varie applicazioni container, i micro-frontend rilasciati verranno caricati all'interno del servizio di archiviazione online Google Cloud Storage, ciascuno nel proprio spazio dedicato.

[14] `https://www.sonatype.com/products/repository-oss`

I BFF prodotti invece, verranno prima containerizzati per poter essere poi collocati nel relativo ambiente di produzione, a questo scopo esistono diversi *container engine* come Docker[15] e Rkt[16]. Docker è lo strumento ufficiale scelto in azienda, le cui immagini verranno poi caricate nel relativo registro aziendale, pronte per essere dispiegate all'interno del servizio Google Kubernets Engine, sempre offerto dal *cloud provider* di riferimento dell'organizzazione.

4.4.2 Distribuzione

Come discusso in sezione 4.4.1, alcuni degli artefatti prodotti verranno pubblicati come immagini docker, i container sono un ottimo modo per distribuire e mettere in esecuzione applicazioni, questo grazie anche alle loro dimensioni ridotte e al fatto che offrono un ambiente di esecuzione completo ed autosufficiente che gli conferisce la capacità di scalare orizzontalmente in maniera efficiente.
Pertanto, le moderne applicazioni complesse possono essere costituite da centinaia o addirittura migliaia di servizi con diverse interdipendenze. All'interno di contesti di questo tipo può essere molto complicato gestire i container che eseguono questi servizi e garantire in ogni momento un corretto funzionamento dell'applicativo.

Per rispondere a questo tipo di problematiche sono nati vari strumenti definiti orchestratori di container. Questi consentono la gestione di architetture basate su questa tecnologia in maniera agile, fornendo un unico punto di accesso centralizzato per organizzare tali container all'interno di cluster, gestirne la scalabilità e l'integrità nel tempo.
Un orchestratore offre in genere le seguenti funzionalità principali: controllo sulle risorse consumate, scheduling, bilanciamento del carico, controllo dello stato, tolleranza agli errori e scalabilità automatica [44].

[15] `https://www.docker.com/`
[16] `https://github.com/rkt/rkt`

Attualmente sono disponibili sul mercato diversi orchestratori di container, tra i più diffusi vi sono:

Kubernetes (K8s) è un software open-source. Inizialmente sviluppato da Google, viene ora mantenuto dalla Cloud Native Computing Foundation[17]. Attualmente vari cloud provider offrono come servizio piattaforme sul quale è presente Kubernetes oppure una loro distribuzione dell'orchestratore [45].

Docker swarm è lo strumento nativo di Docker per la gestione e l'orchestrazione dei cluster. Nato nel 2014 con il nome di Docker Swarm "Classic" standalone[18], è stato archiviato e non è più sviluppato attivamente. Attualmente le versioni recenti di Docker includono la *modalità swarm*[19] integrandola direttamente all'interno del Docker Engine. Tramite essa, gli amministratori e gli sviluppatori di software possono creare e gestire un sistema virtuale noto come *swarm* composto da uno o più nodi, utilizzando direttamente la Docker CLI.

Nomad [20] una soluzione open-source della società di software HashiCorp, è stato progettato per essere semplice e leggero, è un orchestratore flessibile che permette di distribuire e gestire container, applicazioni non incluse in container e macchine virtuali. È disponibile anche una versione commerciale, chiamata Nomad Enterprise.

Marathon [21] è un framework open-source per la gestione e l'orchestrazione dei container basato su Apache Mesos[22], scritto nel linguaggio di programmazione Scala. Marathon è una soluzione completamente basata su REST e può essere utilizzata anche tramite un'interfaccia utente web.

[17] `https://www.cncf.io/`
[18] `https://github.com/docker-archive/classicswarm`
[19] `https://docs.docker.com/engine/swarm/`
[20] `https://www.nomadproject.io/`
[21] `https://mesosphere.github.io/marathon/`
[22] `https://mesos.apache.org/`

L'orchestratore scelto come strumento di riferimento dall'azienda è Kubernetes. Per questa ragione verranno ora approfonditi i suoi aspetti principali e le sue caratteristiche.

Kubernetes

Kubernetes (K8s) è un software open-source, inizialmente sviluppato da Google, attualmente mantenuto dalla Cloud Native Computing Foundation. Permette di gestire in maniera efficace ed efficiente cluster di host, fisici e virtuali, su cui vengono eseguiti container, eliminando i processi manuali tipicamente adottati in fase di rilascio e consentendo di realizzare sistemi dinamici in grado scalare in base alle esigenze.

K8s definisce una serie di astrazioni per rappresentare la struttura del sistema e il suo stato. Queste astrazioni sono definite *Kubernetes Objects*, la loro descrizione viene tipicamente espressa all'interno di file chiamati *manifest* rappresentati in formato *yaml*[23]. K8s adotta quindi un approccio dichiarativo, alternativa ad un più classico approccio imperativo, dove lo stato del sistema è definito dal risultato dell'esecuzione di una serie di istruzioni. Mentre i comandi, espressi quindi in forma imperativa, definiscono le azioni da compiere, la configurazioni espresse in maniera dichiarativa definiscono lo stato finale desiderato del sistema.
Dato che una configurazione dichiarativa risulta di molta più facile comprensione, anche prima di essere applicata, questo approccio è tendenzialmente meno soggetto ad errori. Inoltre, le tecniche tradizionali di sviluppo del software, come il controllo e la validazione del codice sorgente, possono essere applicate all'interno di questo contesto in modi non consentiti a sistemi realizzati con un approccio imperativo, questa proprietà del sistema rende addirittura banale ripristinare lo stato precedente ad una modifica effettuata [46].

[23] `https://yaml.org/`

Kubernetes permette di semplificare la gestione di numerose attività, tra cui [47]:

- *Rilevamento dei servizi e bilanciamento del carico*: grazie a K8s é possibile esporre i container all'esterno del cluster, inoltre gestisce il traffico in ingresso in modo che questo venga distribuito su più istanze in maniera ottimale, garantendo quindi che il servizio rimanga stabile.

- *Orchestrazione dello storage*: K8s permette di assegnare automaticamente ad ogni container in esecuzione sul cluster il sistema di archiviazione preferito, esistono infatti diverse opzioni come storage locale e di rete, dischi forniti da cloud pubblici, e altro ancora.

- *Rilascio e ripristino automatizzati*: tramite K8s é possibile descrivere lo stato desiderato per i propri container, successivamente K8s stesso si occuperà di alterare quello attuale per raggiungere quello desiderato ad una velocità controllata, al fine di garantire il minimo disservizio possibile e nel caso di anomalie durante la procedura si occuperà di ripristinare la situazione di partenza.

- *Ottimizzazione dei carichi*: a K8s viene affidata la gestione del cluster, si occuperà dunque anche di allocare i container sui nodi, in modo da massimizzare l'uso delle risorse a disposizione in funzione di quelle richieste, scalando il numero di istanze in base al carico di lavoro e alle politiche specificate.

- *Autoguarigione*: K8s si occupa di monitorare lo stato dei container, al fine di rilevare e successivamente intervenire in caso di malfunzionamenti, provvedendo a terminare e riavviare le istanze soggette a guasti.

- *Gestione di informazioni sensibili e della configurazione*: K8s consente di memorizzare informazioni sensibili, inoltre la gestione di queste e delle configurazioni può essere compiuta evitando di ricostruire e di distribuire nuovamente i container.

Un cluster Kubernetes è costituito da un insieme di macchine worker, chiamate *nodi*, che eseguono applicazioni containerizzate. Il responsabile della gestione dei nodi worker è il master, chiamato Control Plane. Questo componente è la principale unità di controllo del cluster, reagisce agli eventi, gestisce il carico di lavoro e dirige la comunicazione attraverso il sistema.
Il Control Plane può essere eseguito all'interno di un singolo nodo oppure su più nodi master in modo da fornire un'elevata disponibilità del sistema. I vari elementi che lo compongono sono, il *kube-scheduler* incaricato di gestire l'esecuzione dei container, il *kube-apiserver* che espone l'interfaccia verso l'esterno, *etcd* usato per persistere i dati del cluster nella forma chiave-valore ed il *kube-controller-manager* che esegue i controller in ascolto sugli eventi del cluster.
All'interno di ogni nodo worker è invece presente un *container runtime* responsabile dell'esecuzione dei container, un *kubelet* incaricato della comunicazione con il master ed un *kube-proxy* che si occupa di gestire il traffico del nodo e risolvere quindi i problemi di instradamento.

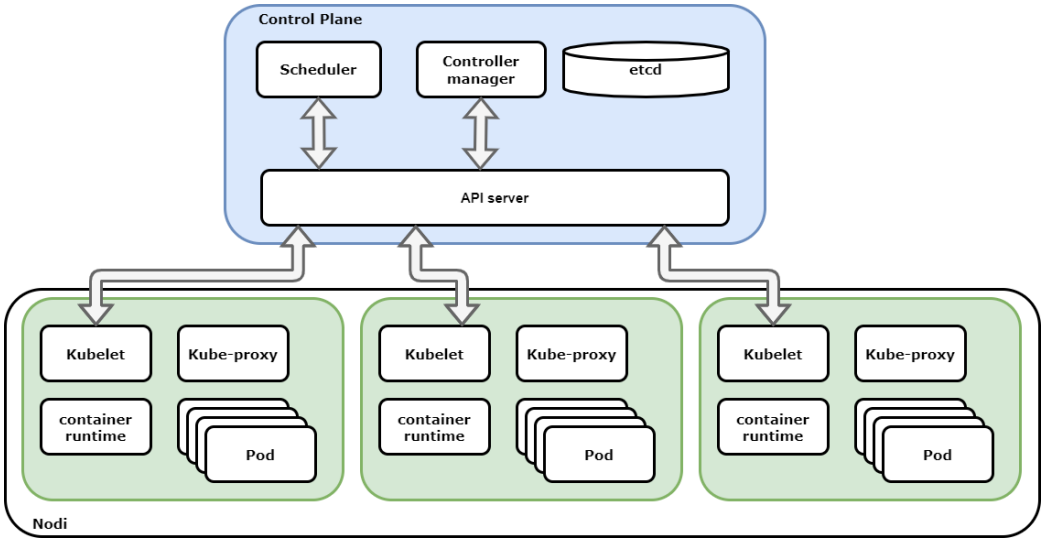

Figura 4.3: Componenti di un cluster Kubernetes.

Tra i principali *Kubernetes Objects* che possiamo trovare all'interno di K8s vi sono i seguenti:

Pod sono le più piccole unità di elaborazione che possono essere istanziate in Kubernetes. Un Pod è costituito da uno o più container allocati sullo stesso nodo, con risorse di rete e storage condivisi.
Essendo atomico un Pod viene eseguito solo su unico nodo, quando viene schedulato vengono eseguiti tutti i suoi container e quando viene distrutto vengono distrutti tutti i container al suo interno.

ReplicaSet contiene al suo interno il descrittore di un Pod e ne definisce il numero delle repliche che devono essere mantenute all'interno del sistema. Nella definizione di un ReplicaSet vengono utilizzati dei selettori, la cui valutazione risulterà nell'identificare tutti i Pod ad esso associati.

Deployment consiste nell'estensione di un ReplicaSet, al quale aggiunge la verifica del corretto funzionamento dell'applicazione in esecuzione dell'interno del Pod e la gestione degli aggiornamenti, questi possono avvenire in modalità *rolling-update*, quindi in maniera graduale e senza interruzione di servizio.

StatefulSet è l'oggetto di K8s utilizzato per gestire le applicazioni stateful. È simile ad un ReplicaSet ma è pensato per quei contesti dove le istanze dell'applicazione devono essere trattate come elementi non fungibili, ognuna con un nome e uno stato stabili.

Job è un oggetto usato per descrivere attività di lavoro finite. I Job sono usati per supervisionare uno o più Pod che eseguono processi per un certo tempo fino al completamento, ad esempio un calcolo o un'operazione di backup.

CronJob sono utili per creare attività periodiche ricorrenti attraverso la creazione di Job in maniera pianificata.

Deamonset assicura che tutti i Nodi del cluster eseguano una copia di un determinato Pod, questo è utile per casi d'uso come la raccolta di log, ingress controller e servizi di archiviazione.

Namespace forniscono un meccanismo per isolare gruppi di risorse all'interno di un singolo cluster.

Service consiste in un modo astratto per esporre un'applicazione in esecuzione su un set di Pod come servizio di rete. Esistono diversi tipi di Service:

ClusterIP visibilità limitata all'interno del cluster.

NodePort visibilità estesa anche all'esterno del cluster.

LoadBalancer estende un NodePort aggiungendo una distribuzione del carico di lavoro.

Volume i filesystem all'interno dei container forniscono un'archiviazione temporanea, questo rappresenta un limite non indifferente in applicazioni non banali. Esistono vari tipi di volumi che consentono un'archiviazione persistente dei dati.

ConfigMap i dati archiviati in questo tipo di volume possono essere referenziati e quindi consumati da applicazioni containerizzate in esecuzione in un Pod.

Secret questo tipo di volume viene utilizzato per passare informazioni sensibili, come password, ai Pod.

Helm

Helm[24] è stato introdotto a partire dalla metà del 2016, come tecnologia per la gestione dei pacchetti e la distribuzione di applicazioni complesse, all'interno di piattaforme cloud basate su Kubernetes [48].

[24] `https://helm.sh/`

Scrivere e gestire i descrittori di Kubernetes per tutti gli oggetti richiesti può essere un'attività noiosa e dispendiosa in termini di tempo. Helm semplifica questo processo e crea un singolo pacchetto condivisibile e pubblicabile all'interno di appositi repository, siano essi pubblici o privati.
Condividere questi elementi all'interno di un'organizzazione o tra organizzazioni differenti permette di ottimizzare gli sforzi, consentendo quindi una maggiore efficienza ed una riduzione della propagazione degli errori.

I pacchetti gestiti da Helm si chiamano *Helm Charts*, sono composti da file di configurazione e modelli espressi in formato *yaml*, quando richiesto Helm andrà a sostituire le variabili di configurazione contenute nei modelli con i valori specificati, per generare i manifest di K8s pronti per l'uso.
Questo processo permette di semplificare notevolmente la gestione di applicazioni che devono essere rilasciate su ambienti diversi. Un esempio di questa gestione è visibile all'interno del progetto *sportello virtuale code*, dove al percorso *charts/sportellovirtuale* è stato definito lo specifico Helm Chart con i modelli e le configurazioni comuni, mentre al percorso *deployments/* sono stati definiti i file *yaml*, contenti le variabili specifiche per ogni ambiente.
In fase di rilascio, sarà quindi necessario fornire al comando *helm upgrade* i parametri desiderati a seconda del contesto, al fine di raggiungere l'obiettivo desiderato, come mostrato nell'esempio seguente:

```
helm upgrade -i sportellovirtuale ./charts/sportellovirtuale
        --create-namespace
        -n sportellovirtuale
        --atomic
        --set image.tag=$CI_COMMIT_SHORT_SHA
        -f deployments/staging.yaml
```

4.5 Infrastructure as Code

Quando si parla di infrastruttura IT, ci si riferisce comunemente ad un elevato numero di componenti, dall'hardware al software, come spazio di ar-

chiviazione, potenza di calcolo o risorse di rete. Gestire un'infrastruttura IT tradizionale è un'attività complessa che richiede tempo, in quanto realizzata in maniera manuale ed in carico ad amministratori di sistema o team specializzati.

In seguito alla diffusione su larga scala del cloud computing, in particolare del modello chiamato Infrastructure as a Service (IaaS), si sono diffuse anche un insieme di pratiche classificate come Infrastructure as Code (IaC). Il concetto IaC viene utilizzato per descrivere l'idea che quasi tutte le azioni eseguite sull'infrastruttura possono essere automatizzate [49]. All'interno di questo paradigma i componenti dell'infrastruttura vengono appunto modellati attraverso la generazione di codice, questo consente di adottare le stesse buone pratiche adottate per la scrittura di qualsiasi tipo di software e di sfruttare gli stessi strumenti.
Linguaggi dedicati, tipicamente di scripting, consentono di definire le caratteristiche dell'ambiente desiderato, come il numero di macchine richieste o la quantità di memoria RAM necessaria. I file risultanti possono essere versionati e modificati come qualsiasi altro tipo di codice sorgente. L'adozione di queste pratiche permette di abbandonare le precedenti attività manuali e di avviare un nuovo ambiente virtuale o di effettuare aggiornamenti in maniera agile [50], dando vita ad infrastrutture mutevoli [51].
Le pratiche di IaC permettono di automatizzare e semplificare il ciclo di rilascio e di test, riducendo contemporaneamente il rischio di errore umano [50].

Le pratiche di IaC si basano su un insieme di concetti principali come:

Ripetibilità La configurazione dell'infrastruttura definita attraverso queste tecniche nasce per essere ripetibile *by design*, in questo modo deve essere possibile giungere allo stato desiderato a partire da qualsiasi stato iniziale del sistema [52].

Ottimizzazione dei costi Automatizzare i processi di gestione dell'infrastruttura permette di risparmiare tempo e denaro che possono essere investiti in altre attività.

Tracciabilità Versionare il codice permette di ricostruire le modifiche apportate alla definizione dell'infrastruttura, fornendo la possibilità di risalire a chi ha compiuto queste modifiche, a cosa è stato cambiato e così via. Questo è fondamentale anche quando è necessario eseguire il rollback ad una versione precedente del codice, magari a seguito del riscontro di eventuali problemi.

Consistenza I sorgenti che definiscono l'architettura sono un'unica fonte di verità, possono essere eseguiti ripetutamente e generare risultati prevedibili ogni volta, inoltre le caratteristiche del sistema vengono espresse in maniera chiara ed inequivocabile. Tutto ciò garantisce che più istanze dello stesso codice di base forniscano un ambiente simile, in questo modo si evitano incoerenze e differenze di configurazione riscontrabili invece durante la creazione manuale di entità infrastrutturali complesse.

Evoluzione continua Nei sistemi tradizionali apportare modifiche è complicato e costoso, pertanto si tende a ridurre al minimo questo genere di attività. Poiché è difficile prevedere come verrà utilizzato un sistema o come i requisiti evolveranno nel tempo, è fondamentale eseguire una fase iniziale di progettazione accurata che tenga conto di tutti i vari possibili scenari futuri, questo porta spesso alla generazione di sistemi più complessi del necessario o con un numero superiore di risorse a disposizione, con conseguenti sprechi.

Grazie alla dinamicità delle infrastrutture cloud basata su IaC è possibile affrontare questo problema semplificando la gestione delle modifiche da apportare, sebbene i sistemi attuali siano progettati per soddisfare i requisiti correnti, le modifiche future devono essere facili da implementare, in modo rapido e sicuro, anche frequentemente.

Efficienza elevata L'automatizzazione dei processi porta ad un'accelerazione drastica del ciclo di rilascio del software e rende il sistema più reattivo garantendo feedback immediati, questo consente di ottimizzare l'intero processo di sviluppo.

All'interno del panorama IaC esistono vari strumenti largamente diffusi, tra questi troviamo Chef, Puppet, Terraform, Ansible, Packer, SaltStack e molti altri. La decisione, presa a livello aziendale, è stata quella di affidarsi a Terraform[25], in particolare alla sua versione open-source, come strumento di riferimento per i progetti futuri.

Terraform

Terraform (TF) è uno strumento, scritto in linguaggio GO[26], rilasciato per la prima volta nel 2014 da HashiCorp[27]. È disponibile in tre diverse edizioni Terraform Open Source, Terraform Cloud (TFC) e Terraform Enterprise (TFE) [53]. Nella sua versione open-source, tramite la Terraform CLI, è possibile gestire centinaia di servizi cloud, infatti il supporto verso molteplici provider è uno dei principali punti di forza del prodotto.
Ricercando all'interno dell'ampio catalogo di plugin messo a disposizione, gli utenti possono aggiungere alle proprie configurazioni il necessario per integrare le piattaforme e i servizi scelti, tra questi possiamo trovare Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, Oracle Cloud Infrastructure ed Alibaba Cloud.
Attraverso l'utilizzo di questi plugin, Terraform permette all'utilizzatore di svincolarsi dalle interfacce pubbliche definite dai singoli cloud provider, fornendo quindi un ottimo livello di astrazione.

Seguendo un approccio dichiarativo, l'infrastruttura finale viene descritta mediante l'utilizzo di una serie di Configuration Files, questi file con esten-

[25] `https://www.terraform.io/`
[26] `https://golang.org/`
[27] `https://www.hashicorp.com/`

sione *.tf* e *tf.json*, permettono di dichiarare l'insieme delle risorse che costituiscono il sistema tramite l'uso di un linguaggio proprietario, chiamato HashiCorp configuration language (HCL). HCL è ampiamente apprezzato per la sua semplicità di utilizzo nonostante la complessità delle tecnologie dell'infrastruttura di destinazione [51], in alternativa è consentito scrivere file di configurazione in formato JSON, la decisione viene demandata all'utente finale.

In seguito alla creazione dell'infrastruttura e ad ogni esecuzione successiva, Terraform registra le informazioni sul sistema all'interno di un file chiamato *Terraform State file*, con estensione *.tfstate*.
Ogni volta che sarà necessario apportare una modifica all'infrastruttura, TF valuterà le differenze tra lo stato attuale della stessa e quello desiderato, quest'ultimo espresso dai file di configurazione opportunamente modificati, al termine di questa elaborazione produrrà un'anteprima delle operazioni necessarie per aggiornare l'infrastruttura.
Giunti a questa fase sarà possibile procedere con l'esecuzione pianificata, in modo che vengano apportate le modifiche necessarie e l'infrastruttura raggiunga effettivamente lo stato richiesto.

4.6 Pipeline

All'interno di questo capitolo sono stati presentati i passi che compongono il processo di rilascio automatizzato, relativamente ai componenti del progetto in esame che verranno realizzati.
In questa sezione verrà illustrata la pipeline di CI/CD nel suo insieme ed il risultato finale che s'intende raggiungere, inizialmente saranno presentati i concetti principali alla base del sistema che si vuole realizzare mentre in secondo luogo lo specifico strumento adottato.

Una pipeline CI/CD si suddivide in sottoinsiemi distinti di attività, ciascuna delle quali contribuisce al raggiungimento dell'obiettivo finale, portan-

do quindi dal codice sorgente sino al rilascio del sistema in produzione, il tutto all'interno di un processo automatizzato che combina i principi di integrazione e distribuzione continua.
Adottare questo genere di strategia consente di ottenere diversi vantaggi, tra cui velocizzare le attività, ridurre gli errori umani, diminuire i costi e garantire che gli standard imposti siano stati soddisfatti.
Un uso frequente è quello di adottare un approccio chiamato *pipeline as code*, il quale consiste nel definire la pipeline in maniera formale, seguendo quelli che sono i principi legati alla infrastructure as code, descritti in sezione 4.5, mantenendo quindi tutti i modelli e le informazioni collegate al processo di rilascio all'interno di un unico file versionato [54].

In generale non esiste un'unica formulazione che possa racchiudere la struttura e le fasi di una pipeline in maniera comune e assoluta, ciò è dato dal fatto che le caratteristiche di questa variano fortemente in base al contesto e alle esigenze specifiche, necessità del team, tipologia di infrastruttura, tecnologie adottate, sono tutti fattori che contribuiscono a definire gli aspetti della pipeline finale.
Per questo motivo all'interno del capitolo sono stati affrontati i passaggi principali del ciclo di rilascio, senza l'obiettivo di fornire una guida che potesse raccogliere tutte le possibili esigenze in senso assoluto, in figura 4.4 viene quindi presentata a titolo di esempio una pipeline generica.

All'interno di progetti caratterizzati da una dimensione importante, come descritto in sezione 1.1, capita spesso di assistere ad una organizzazione che prevede il coinvolgimento di più team, impegnati a compiere le proprie attività operando sulla stessa code base contemporaneamente, condividendo quindi tra loro anche quello che è il proprio ciclo di rilascio.
Organizzare il lavoro secondo un flusso di questo tipo porta all'instaurazione di una serie di dipendenze tra i team in fase di rilascio, questo limite, classico di un approccio monolitico, può insorgere anche nel caso di architetture a micro-frontend con integrazione in fase di compilazione, come descritto

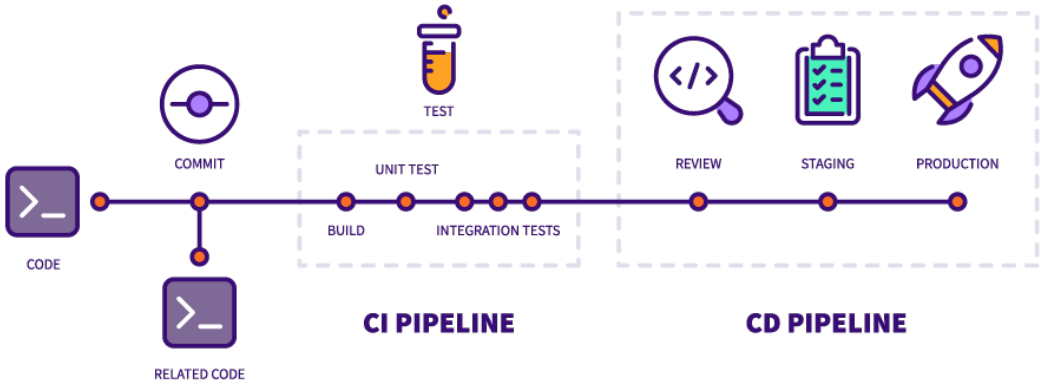

Figura 4.4: Esempio di una generica pipeline CI/CD [55].

in sezione 1.4.1. In figura 4.5 viene illustrato un flusso di lavoro di questo tipo, caratterizzato appunto dal fatto che ogni qual volta che anche solo pochi micro-frontend devono essere aggiornati, l'intera applicazione deve essere ricostruita e ridistribuita.

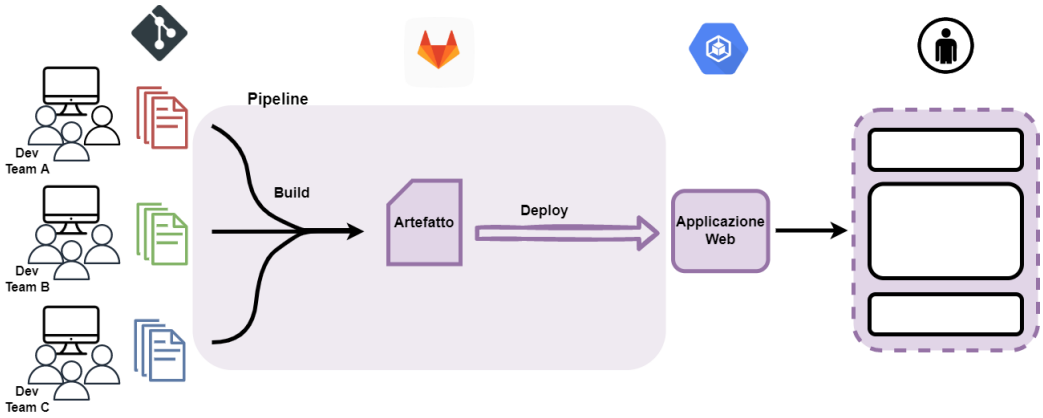

Figura 4.5: Flusso di lavoro caratteristico di una architettura monolitica.

L'approccio che s'intende adottare nel progetto corrente mira ad ottenere il massimo vantaggio che una architettura a micro-frontend può offrire, realizzando un flusso di lavoro nel quale ogni team può lavorare in maniera autonoma, con il massimo disaccoppiamento non solo in fase di sviluppo ma

anche di distribuzione, come rappresentato in figura 4.6.
L'obiettivo dunque è quello di realizzare un sistema composto da diverse pipeline di rilascio indipendenti, associate a ciascun componente e basate su un modello non bloccante [34], ossia senza che si verifichino inutili attese in fase di rilascio dovute da fattori esterni fuori dal controllo del team.
I team potranno quindi svolgere le proprie mansioni in autonomia, dunque operare su repository separati, possedere il controllo sulla pipeline di CI/CD e acquistare la libertà di effettuare rilasci nei tempi e nei modi che ritengono più opportuni. Il processo di lavoro diventerà quindi più agile ed efficiente, in grado di produrre nuove funzionalità attraverso un flusso costante e senza ostacoli [34].

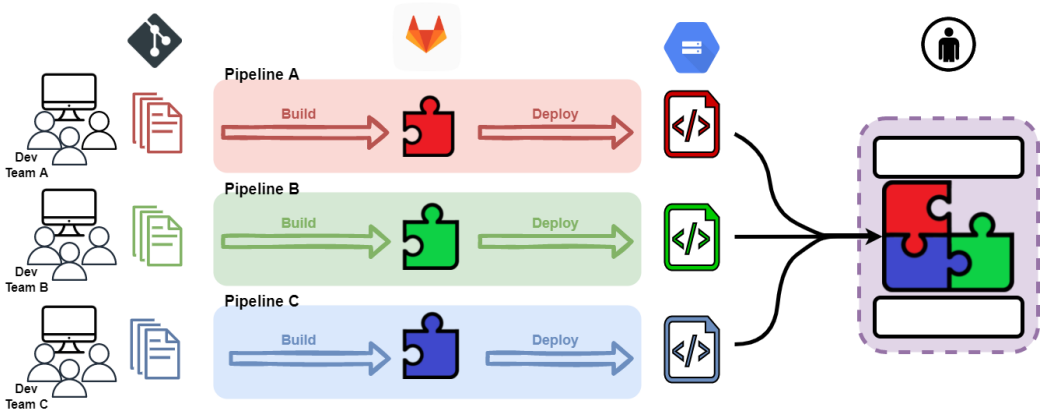

Figura 4.6: Flusso di lavoro caratteristico basato sul massimo grado di indipendenza, proprio di una architettura a micro-frontend.

Come strumento ufficiale per l'implementazione delle pipeline di progetto verranno utilizzate le funzionalità messe a disposizione da GitLab, la scelta è dettata da una direttiva aziendale ed è dovuta anche dal fatto che, come riportato in sezione 4.2, questo strumento è già la piattaforma di controllo di versione di riferimento.
Nato inizialmente semplicemente come un Web-based git repository manager, GitLab si è evoluto ampliando la propria gamma di servizi in modo da

coprire l'intero ciclo di vita dello sviluppo del software, il cui processo di rilascio è definito in maniera dichiarativa con un approccio pipeline as code [54], all'interno di un file *.gitlab-ci.yml* posto nella radice del repository di progetto.
Dentro al file di configurazione possono quindi essere dichiarati i processi, chiamati *job*, contenenti le istruzioni riguardanti ciò che deve essere svolto, compresi gli eventi scatenanti che ne determinano l'avvio dell'esecuzione. In relazione al caso di studio corrente e a quanto definito in sezione 4.1.4, l'aggiornamento dell'ambiente di staging sarà innescato in seguito ad una nuova commit sul ramo master, mentre l'ambiente di produzione verrà aggiornato a seguito del rilascio di una nuova versione del software.
Gli elementi preposti all'esecuzione dei job in attesa vengono chiamati Gitlab Runner, grazie ad essi i processi definiti possono essere svolti in maniera isolata all'interno di un ambiente dedicato, inizializzato con le variabili dinamiche necessarie.
I job da svolgere vengono dichiarati all'interno di quelli che sono chiamati *stage*, ossia le fasi della pipeline identificate da una parola chiave come per esempio build e deploy, questo determina l'ordine in cui verranno svolte le attività, infatti nelle forma più elementare di configurazione, tutti gli elementi appartenenti allo stessa fase vengono eseguiti in parallelo, solo una volta che tutti saranno completati comincerà l'esecuzione delle attività appartenenti allo stage successivo [56].

Capitolo 5

Prototipi realizzati

Al fine di verificare la fattibilità del progetto sono state compiute più indagini e sviluppati vari proof of concept, in questo capitolo verranno descritti i più recenti prototipi realizzati. Questi sono stati compiuti allo scopo di effettuare uno studio pratico a supporto delle fasi di analisi future, per raccogliere le conoscenze necessarie al fine di adottare le giuste decisioni e per verificare la fattibilità delle soluzioni ideate.
I componenti scelti come progetti pilota riguardano due nuove funzionalità e non sono quindi servizi già offerti all'interno degli applicativi del gruppo, sono stati selezionati per la loro limitata complessità, questo in modo da raggiungere i risultati cercati nel minor tempo possibile.

I due progetti che verranno presentati sono quindi Sportello Virtuale Code e Calcolatrice TARI, del quale verranno descritte le caratteristiche principali, il funzionamento e le considerazioni finali, riguardanti i limiti dei prototipi realizzati sul quale investire per ottenere risultati migliori.
All'interno del capitolo verranno anche riportati gli attuali responsabili dei due progetti, al quale hanno attivamente partecipato tutti e tre i team del gruppo, infatti anche per testare la futura modalità di lavoro i due prototipi sono stati realizzati in parallelo e assegnati a diversi membri del gruppo.
I BFF sono stati affidati al team di J-City Gov, in quanto al momento in possesso di maggiori competenze in ambito Java, invece per quanto riguarda

i micro-frontend, la Calcolatrice TARI è stata inizialmente presa in carico dal team Sportello Telematico mentre al team di J-City Gov è stato assegnato il progetto Sportello Virtuale Code, il team di Municipium ha invece fornito supporto ad entrambi nelle fasi iniziali dello sviluppo, vista la grande esperienza con il framework Angular.

A seguito dell'analisi dei dati raccolti, tramite i due progetti pilota, verranno presentati gli aspetti finali collegati ed illustrate le misure ideate per affinare la soluzione finale.

5.1 Sportello Virtuale Code

Lo Sportello Virtuale[1] è un servizio di Maggioli ideato per consentire agli enti locali di erogare le normali attività di sportello in maniera digitale. Mediante questo servizio gli operatori del comune possono quindi ricevere il pubblico in videoconferenza, dove è possibile scambiare documenti in formato elettronico e dare indicazioni all'utente, anche mediante condivisione dello schermo, rispetto a dove trovare informazioni e moduli all'interno del sito istituzionale.
Il servizio è fruibile in due formati diversi, sia in modalità ad accesso libero, sia tramite appuntamento prenotabile attraverso la piattaforma stessa. Il progetto Sportello Virtuale Code riguarda esclusivamente la prima delle due modalità indicate ed il suo compito è limitato alla gestione degli utenti in coda e non all'erogazione del servizio di videoconferenza che viene invece offerto da un applicativo differente.

Lo Sportello Virtuale Code, attualmente in carico al team che si occupa di J-City Gov, è stato strutturato come definito in fase di analisi e riassunto in sezione 3.6.3, è quindi composto da un micro-frontend e dal relativo BFF, quest'ultimo incaricato di fornire la giusta astrazione rispetto ai servizi di

[1] `https://www.maggioli.com/it-it/settori/pubblica-amministrazione/relazioni-con-il-pubblico-e-servizi-per-il-cittadino`

back-office, in questo caso offerti da Sicr@web, in particolare contenuti nel modulo J-Iride.

Nel seguito verranno presentate le sue caratteristiche e le considerazioni nate durante il suo sviluppo, relative all'intero progetto di integrazione e migrazione.

5.1.1 Operatività

L'operatività del componente sviluppato è piuttosto semplice e lineare, proprio per questo motivo è stato scelto come primo caso di studio del nuovo sistema. Nel seguito verranno presentati i passaggi principali, i quali sono rappresentati in figura 5.1 e 5.2, dove viene illustrato il normale flusso di lavoro legato al servizio.

Lo scenario d'uso da immaginare è quello di un comune cittadino alle prese con un qualsiasi servizio digitale dell'ente, servizi tendenzialmente consumati da utenti occasionali, supponendo che si trovi in difficoltà nel raggiungere il proprio obiettivo, decide quindi di avvalersi del supporto di un operatore di sportello per ricevere aiuto in merito.
Il servizio, fornito tramite un link diretto, dovrà quindi essere opportunamente esposto agli utenti della piattaforma, a questo punto l'utente, non necessariamente autenticato al sistema, atterrerà nella schermata iniziale dello Sportello Virtuale Code dove potrà scegliere a quale sportello rivolgersi in base alle proprie esigenze.

L'elenco degli sportelli a disposizione viene recuperato dinamicamente ad ogni accesso interrogando i servizi offerti da J-Iride, oltre a questi vengono recuperate anche le informazioni collegate a ciascuno di essi, come gli orari di apertura e chiusura oppure il numero di utenti attualmente in coda. Il cittadino, una volta effettuata la sottoscrizione ad un determinato sportello, potrà annullare la richiesta in un qualsiasi momento per tornare alla schermata iniziale oppure rimanere in attesa fino al raggiungimento del proprio turno.

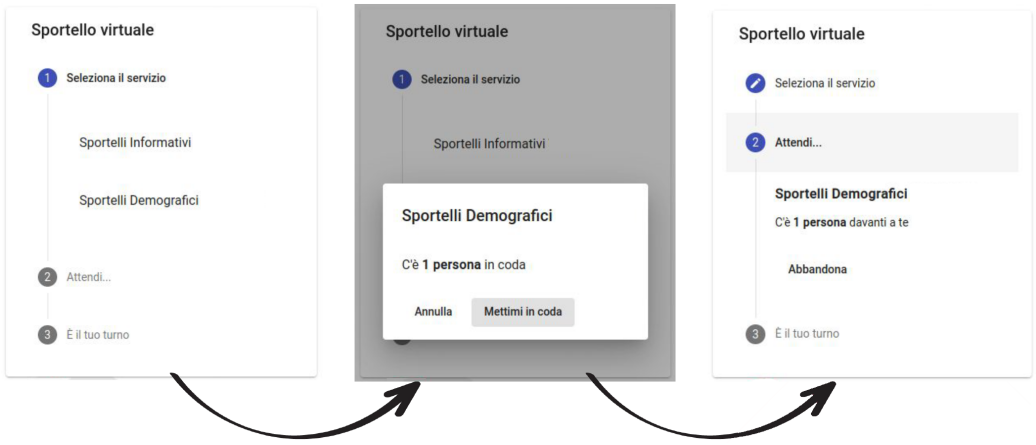

Figura 5.1: Illustrazione della fase iniziale del flusso di lavoro dello Sportello Virtuale Code.

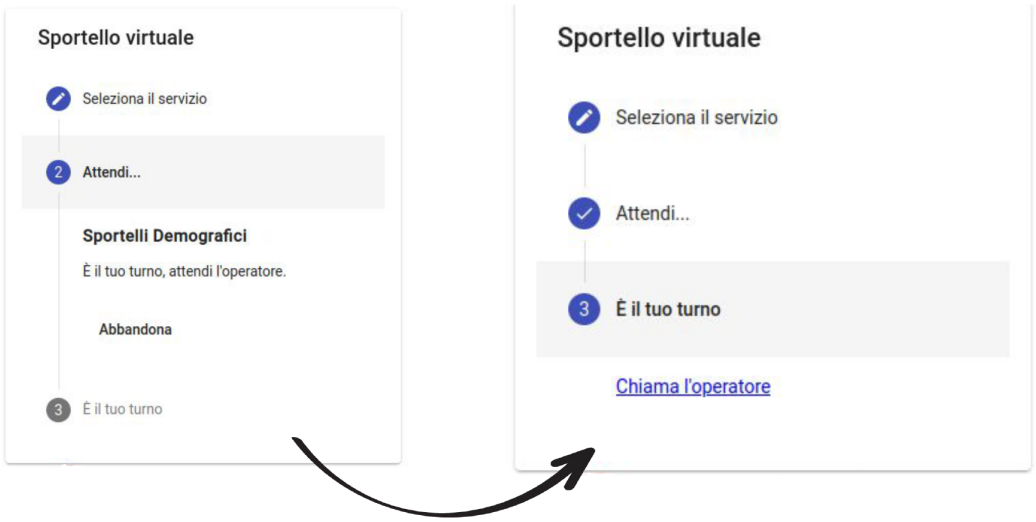

Figura 5.2: Illustrazione della fase finale del flusso di lavoro dello Sportello Virtuale Code.

Arrivati a questo punto, l'operatore comunale potrà accettare in carico la richiesta attraverso Sicr@web, questa azione innesca una serie di operazioni automatiche che porteranno alla creazione di un apposito spazio all'interno del quale potrà essere effettuata la videoconferenza, la partecipazione dell'u-

tente avverrà grazie al collegamento che gli sarà presentato, tramite il quale potrà accedere e ricevere quindi l'assistenza richiesta.

5.1.2 Considerazioni

Un primo rilascio del componente Sportello Virtuale Code è già stato realizzato con successo, successivamente l'inclusione di questo all'interno di ciascuno dei tre principali prodotti del team dei Servizi al Cittadino, ha permesso di effettuare delle importanti valutazioni.

I risultati ottenuti hanno portato alla luce un limite non trascurabile di questa gestione, preannunciato e descritto già in sezione 3.4.2, ossia la necessità di un elevato livello di isolamento, in questo caso in particolare tra i micro-frontend e l'applicazione container.
Questi primi test hanno infatti evidenziato come la situazione attuale sia da ritenersi insoddisfacente per gli standard attesi, il tutto per via delle influenze reciproche, potenzialmente evidenti e non trascurabili, osservate tra questo primo componente e i vari prodotti all'interno del quale è stato incluso.
Benché non siano da considerarsi problematiche per il micro-frontend corrente, in quanto minime e non facilmente individuabili, il fenomeno non deve essere sottovalutato, in particolare in relazione ai progetti futuri, caratterizzati da un maggiore livello di complessità rispetto al componente Sportello Virtuale Code.

Il problema dunque, anche se non immediato, dovrà essere opportunamente gestito, questo anche nell'ottica di coinvolgere ulteriori team aziendali, esterni al gruppo, nello sviluppo dei micro-frontend futuri.
Un livello di isolamento non adeguato infatti, comporterebbe non poche problematiche all'interno del contesto corrente, soprattutto a livello organizzativo e gestionale, visto in particolare la complessità aggiuntiva a cui finirebbe inevitabilmente per condurre in fase di collaudo.

5.2 Calcolatrice TARI

Il servizio è stato pensato per consentire ai cittadini di calcolare la tassa sui rifiuti, conosciuta come TARI, riferita alle proprie utenze in modo personalizzato. La TARI, posta a carico dell'utilizzatore, è il tributo destinato a finanziare i costi del servizio di raccolta e smaltimento dei rifiuti, è stata introdotta il 27 dicembre 2013 e a decorrere dal 2014 ha sostituito il tributo comunale sui rifiuti e sui servizi (TARES).

Il servizio ideato da Maggioli, attualmente in sviluppo, prevede una realizzazione basata su più fasi. Nella prima versione la funzionalità sarà limitata al calcolo dell'importo dovuto, il saldo ottenuto verrà espresso in soluzione unica senza rateizzazione, attraverso il sistema non sarà però possibile effettuarne il pagamento, in quanto il servizio è pensato per operare all'interno di un contesto in regime di liquidazione d'ufficio, ossia in una situazione in cui sarà l'ente stesso a comunicare al contribuente, tramite apposito documento, modalità e importo dovuto.
A partire dalla seconda versione verrà supportata anche la funzione di autoliquidazione. L'operatività del sistema sarà la stessa della versione precedente, la differenza riguarderà il calcolo finale, questo non verrà presentato esclusivamente in soluzione unica ma anche in forma rateizzata accompagnato da relative scadenze, il pagamento potrà essere effettuato tramite modello F24 oppure sfruttando pagoPA[2].

Il progetto Calcolatrice TARI è attualmente in carico al team di J-City Gov, anche questo, come lo Sportello Virtuale Code, è stato strutturato rispettando quanto definito in fase di analisi, è quindi composto da un micro-frontend e dal relativo BFF, i servizi di back-office sono nuovamente offerti da Sicr@web, questa volta però sono contenuti all'interno del modulo J-Trib e sviluppati quindi dal relativo team aziendale.

[2] `https://www.pagopa.gov.it/`

Nel seguito vengono riportate le caratteristiche operative del componente, al termine sono elencate le considerazioni che è stato possibile trarre durante lo sviluppo del servizio.

5.2.1 Operatività

Il componente in questione comporta un coefficiente di difficoltà maggiore dal punto di vista tecnico rispetto allo Sportello Virtuale Code, tuttavia i requisiti del sistema, limitatamente a quanto riguarda la prima versione dell'applicativo, sono stati ritenuti non eccessivamente complessi, per questo motivo il componente è stato considerato idoneo, assieme al precedente, come caso di studio iniziale della futura architettura.

Come descritto il componente è stato ideato per permettere ai cittadini di effettuare il calcolo relativo alla tassa sui rifiuti. Il flusso di lavoro collegato al servizio parte dunque dalla scelta dell'anno di riferimento, successivamente permette di procedere con l'inserimento dei dati riguardanti le utenze assegnate, queste comprendono locali o aree scoperte suscettibili di produrre rifiuti, siano esse in possesso o detenute a qualsiasi titolo dal contribuente.
La sezione relativa alla raccolta delle informazioni sull'utenza, come riportato in figura 5.3 e 5.4, si suddivide quindi su due differenti casistiche che richiedono una differente gestione di calcolo:

- *utenza domestica*: immobili ad uso privato. In questo caso il calcolo deve essere espresso anche in funzione del numero di occupanti e della tipologia di pertinenza in oggetto, devono infatti essere conteggiate anche le pertinenze relative all'abitazione, siano esse coperte o scoperte, come autorimesse, cantine, depositi e garage.

- *utenza non domestica*: immobili utilizzati per attività produttive, commerciali ed enti. Questa casistica richiede che venga specificata la tipologia di sottocategoria collegata all'immobile, questo al fine di conoscere il coefficiente di produttività rifiuti specifico per la categoria in cui l'attività in questione si colloca.

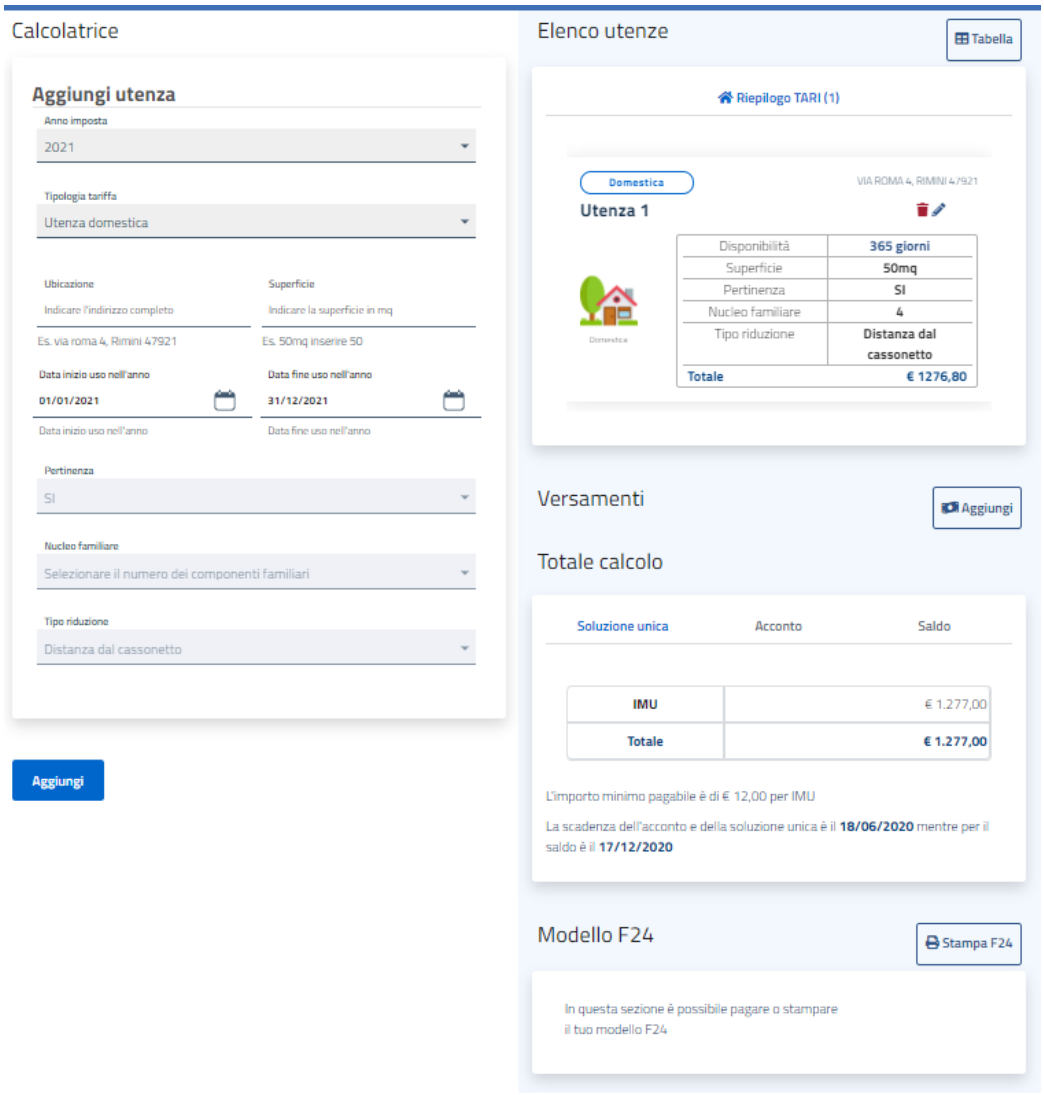

Figura 5.3: Mockup del componente Calcolatrice TARI nel caso di utenza domestica.

In aggiunta alle informazioni specifiche per tipologia di utenza, devono essere richieste anche un'insieme di informazioni comuni, quali la superficie, l'indirizzo, il periodo di possesso ed eventualmente il tipo di riduzione da applicare al calcolo.

Una volta completato l'inserimento delle informazioni relative alla singola

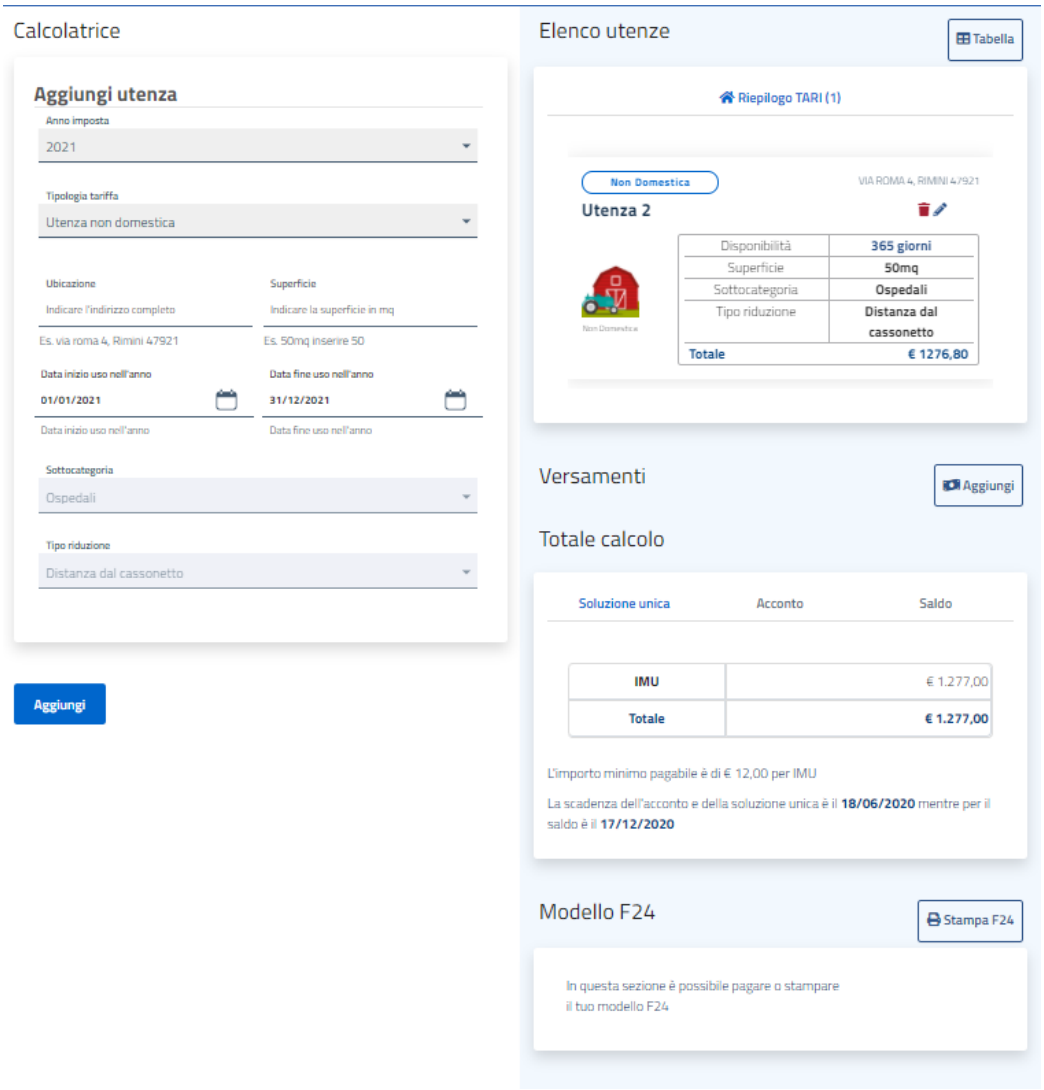

Figura 5.4: Mockup del componente Calcolatrice TARI nel caso di utenza non domestica.

utenza, deve essere possibile per l'utente modificare o cancellare utenze già inserite oppure aggiungerne di nuove. Le informazioni riassuntive, inclusive dei parziali di calcolo delle singoli voci espresse, vengono proposte all'utente tramite apposita sezione, convenientemente accompagnate dal riepilogo dei dati inseriti.

Il totale dell'importo dovuto viene riportato in funzione delle utenze inserite e scontato opportunamente, nel caso cui l'utente specifichi eventuali somme già versate.

Infine, nel caso di supporto alla funzione di autoliquidazione, all'utente viene data la possibilità di effettuare il pagamento dell'importo calcolato, riportato in forma rateizzata accompagnato da relative scadenze, le modalità di pagamento supportate, come anticipato, saranno attraverso la stampa del modello F24 oppure con l'ausilio del servizio pagoPA.

5.2.2 Considerazioni

A differenza del progetto Sportello Virtuale Code, attualmente lo sviluppo del componente Calcolatrice TARI non è ancora stato ultimato, le attività di evoluzione del micro-frontend, in origine in mano ai membri del team di Sportello Telematico, sono state trasferite ai membri del team di J-City Gov. Questo si è reso necessario a seguito della riassegnazione di alcune risorse del team iniziale, il tutto dovuto alla necessità di dover far fronte ad una serie di esigenze emerse sull'applicativo principale in gestione e ad altre dinamiche relative al personale, collegate all'uscita dal team di un membro chiave del gruppo, in precedenza assegnato allo sviluppo del micro-frontend.

Il fatto di aver trasferito la gestione del micro-frontend, in mano allo stesso team responsabile del progetto Sportello Virtuale Code, ha permesso di portare alla luce in maniera anticipata il fatto che i due micro-frontend, il cui sviluppo fino a quel momento era stato portato avanti in maniera autonoma e parallela, fornissero una esperienza utente distante l'uno dall'altro.
Questo fenomeno è il sintomo di una non adeguata gestione di uno dei possibili svantaggi legati all'adozione dell'architettura scelta, già analizzato in sezione 1.3, ossia della mancanza di una UX coerente e consistente.

La realizzazione di due interfacce dallo stile molto diverso, è scaturita dal fatto che i prototipi dei due micro-frontend siano stati sviluppati con

tecnologie nuove, dove gli sviluppatori non erano dotati di grande esperienza, unito alla mancanza di comunicazione tra i due team e all'assenza di linee guida chiare in merito, il tutto dovuto allo stato embrionale del progetto nel momento in cui il loro sviluppo è cominciato.
La causa, in particolare, è principalmente correlata al fatto che per la loro realizzazione sono state adottate librerie di componenti UI differenti, nel primo caso infatti è stato fatto uso della libreria Angular Material[3] mentre nel secondo di ngx-bootstrap[4] che si basa sulla versione 4 del framework Bootstrap.

Una forte unità di stile tra i micro-frontend e il fatto di seguire le direttive AgID in maniera appropriata, rappresentano un vincolo fondamentale del progetto. Per questo motivo lo sviluppo del prototipo legato alla Calcolatrice TARI è stato temporaneamente sospeso, questo visto anche la più che sufficiente mole di informazioni raccolta fino al momento attraverso i due casi di studio.
Le attività riprenderanno al termine dell'analisi del problema e a seguito della definizione di una soluzione stabile all'esigenza individuata.

5.3 Aspetti finali

In questa sezione verranno descritti quelli che sono gli aspetti conclusivi inerenti al progetto raccolti sino ad ora, questo a seguito delle esperienze maturate attraverso lo studio dei componenti Sportello Virtuale Code e Calcolatrice TARI.
Saranno descritti più nel dettaglio i particolari legati all'integrazione del lavoro effettuato e le sue modalità in relazione ai prodotti aziendali, successivamente verranno presentate le soluzioni individuate dal team per rispondere a quelle che sono le problematiche, in particolare in materia di presentazione, emerse durante lo sviluppo di questi due primi casi di studio.

[3] `https://material.angular.io/`
[4] `https://valor-software.com/ngx-bootstrap`

Lo scopo di questi adattamenti sarà quello di arrivare ad ottimizzare il modello di sviluppo, al fine di migliorare la struttura dei singoli componenti nonché la qualità dell'intero sistema, questo verrà effettuato grazie alle nozioni apprese, discusse nei precedenti capitoli e che sono quindi legate direttamente alle architetture a micro-frontend.

5.3.1 Integrazione

In sezione 3.5.3 è stata dichiarata la scelta di affidarsi ad una strategia di composizione basata su Web Component e le motivazioni collegate ad essa, mentre in sezione 3.6.2, è stato presentato lo stack tecnologico di riferimento ed in particolare la decisione presa in merito al framework JavaScript da adottare per lo sviluppo dei micro-frontend, ossia Angular.
Nella stessa occasione è stata presentata la funzionalità Angular Element, di seguito verrà descritto l'uso della tecnologia all'interno del progetto per consentire l'integrazione dei micro-frontend nei prodotti esistenti, il tutto comincia ovviamente dalla dichiarazione del componente, come mostrato nel listato 5.1.

```
1 @Component({
2   selector: 'sportello-code',
3   templateUrl: './sportello-code.component.html',
4   styleUrls: ['./sportello-code.component.scss']
5 })
6
7 export class SportelloCodeComponent implements OnInit {
8     ...
9 }
```

Listato 5.1: Esempio di dichiarazione di un componente Angular.

Al fine di consentire la conversione di un componente Angular, insieme alle sue dipendenze, in Custom Element, Angular fornisce la funzione mo-

strata nel listato 5.2.

```
createCustomElement<P>( t: Type<any>,
                        c: NgElementConfig
                        ): NgElementConstructor<P>
```

Listato 5.2: Metodo per convertire un generico componente Angular in un Custom Element.

Questa crea una classe che incapsula le funzionalità del componente fornito, inoltre recupera gli attributi di input e di output e li converte, utilizzando l'apposita API, negli attributi del Custom Element.
Il risultato restituito è un costruttore configurato per generare in maniera autonoma istanze del componente e che implementa l'interfaccia mostrata nel listato 5.3.

```
interface NgElementConstructor<P> {
  observedAttributes: string[]
  new (injector?: Injector): NgElement & WithProperties<P>
}
```

Listato 5.3: Interfaccia implementata dal costruttore utilizzato per la registrazione del Custom Element specifico.

Una volta eseguito questo passaggio è possibile procedere come già descritto per un generico Web Component in sezione 3.5.3, ossia richiamando il metodo `customElements.define` per registrare il componente all'interno del `CustomElementRegistry` del browser, in questo modo, ad ogni occorrenza del tag dichiarato, verrà eseguito il costruttore specificato e sarà quindi generata una nuova istanza del Custom Element, l'esempio completo viene mostrato nel listato 5.4.

```
export class SportelloCodeModule {

  constructor(private injector: Injector) {}
```

```
4
5   ngDoBootstrap(appRef: ApplicationRef): void {
6     const el = createCustomElement(SportelloCodeComponent,{
7       injector: this.injector,
8     });
9     customElements.define('sportello-code', el);
10  }
11 }
```

Listato 5.4: Esempio di registrazione di un Angular Element.

La procedura descritta, a seguito anche di quanto realizzato e riportato in sezione 4.3.1, permette al componente di essere facilmente integrato all'interno di ciascun portale attraverso l'inclusione di poche righe di codice, come mostrato nel listato 5.5.

```
1 <sportello-code url="..." ente="Z000"></sportello-code>
2 <script src="https://.../sportello-code.js"></script>
3 <link rel="stylesheet" href="https://.../styles.css" />
4 <link rel="stylesheet" href="https://fonts..." />
```

Listato 5.5: Istruzioni per includere in pagina il componente Sportello Virtuale Code.

Il risultato finale viene riportato in figura 5.5, dove viene mostrato lo Sportello Virtuale Code integrato con successo all'interno di ciascuno dei tre applicativi di riferimento.

5.3.2 Isolamento

Lo studio del componente Sportello Virtuale Code ha portato all'individuazione di un'importante problematica collegata al progetto, descritta in sezione 5.1.2, che consiste in un livello di isolamento non adeguato tra applicazione container e micro-frontend.

I lavori svolti per uniformare i principali applicativi del gruppo Servizi al cittadino, come riportato in sezione 3.4.1, hanno portato anche ad una

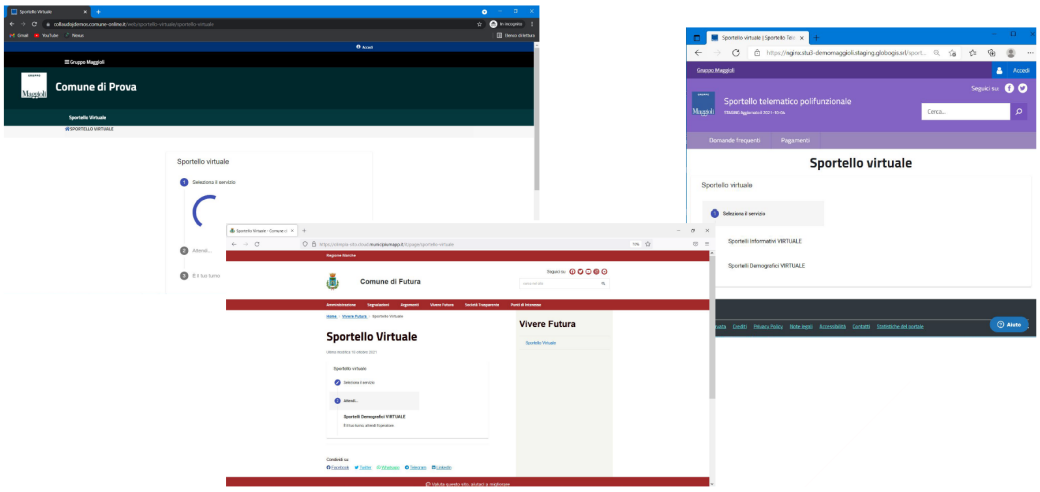

Figura 5.5: Illustrazione dell'integrazione del componente Sportello Virtuale Code all'interno dei tre prodotti aziendali.

omogeneizzazione dello stile, in linea con quelle che sono le direttive definite da AgID. I tre applicativi però hanno raggiunto questo traguardo attraverso l'adattamento delle proprie interfacce in maniera indipendente, senza quindi la condivisione di codice o librerie. Benché presentino quindi un aspetto molto simile, i tre prodotti sono da considerarsi completamente slegati tra loro, questo li rende tre contesti di esecuzione completamente differenti per i componenti che dovranno essere inclusi in essi.
In una situazione di questo tipo garantire un isolamento adeguato rispetto al contesto di esecuzione risulta essenziale, specialmente per ridurre le difficoltà collegate allo sviluppo e al collaudo dei micro-frontend, attività che possono rivelarsi estremamente complesse e richiedere un dispendio di tempo e di risorse notevoli, soprattutto in funzione anche del fatto che saranno team diversi ad occuparsene.

Al fine di introdurre una separazione tra gli stili definiti per ciascuno degli elementi presenti in pagina, in sezione 1.4.1, sono già state citate due possibili strategie applicabili alla metodologia di composizione scelta. La prima consiste nell'adottare una serie di convenzioni durante l'attribuzione dei nomi,

il che permette di limitare le regole di stile definite esclusivamente all'ambito specifico del componente per il quale sono state concepite.
Un differente approccio consiste nell'adottare una tecnica le cui caratteristiche sono state approfondite in sezione 3.5.3, ossia Shadow DOM.
Il framework scelto per lo sviluppo dei micro-frontend consente nativamente di applicare entrambi gli approcci.

In Angular, è possibile incapsulare lo stile definito per un componente all'interno dell'elemento che lo contiene, in modo che non influisca sul resto dell'applicazione.
Il decorator `Component` permette di convertire una classe in un componente Angular, infatti consente di dichiarare i metadati di configurazione che determinano come il componente deve essere elaborato, istanziato e utilizzato in fase di esecuzione. Inoltre permette di specificare l'opzione `encapsulation`[5] che consente di definire come l'incapsulamento dovrà essere applicato, il che avviene tramite uno dei seguenti valori:

- `ViewEncapsulation.ShadowDom`, tramite l'uso dell'API Shadow DOM, integrata nel browser, il componente viene incapsulato all'interno di uno *ShadowRoot*, di conseguenza gli stili forniti vengono applicati esclusivamente ad esso senza influenze reciproche con il Light DOM.

- `ViewEncapsulation.Emulated`, i selettori CSS del componente vengono alterati in modo da essere applicati solo alla vista dello stesso ed evitare che influenzino gli altri elementi nell'applicazione. Questa è la modalità consigliata ufficialmente e utilizzata come default.

- `ViewEncapsulation.None`, non viene applicato alcun tipo di incapsulamento del componente, il che significa che qualsiasi stile specificato per esso viene effettivamente applicato globalmente e può influenzare qualsiasi elemento HTML presente all'interno dell'applicazione.

[5] `https://angular.io/guide/view-encapsulation`

Gli studi fatti a seguito dello sviluppo del componente Sportello Virtuale Code, hanno permesso di verificare con mano il livello di isolamento conseguente alla gestione di default suggerita del framework, fondata come descritto sull'adozione di particolari politiche e manipolazioni dei nomi ad opera di Angular.
I risultati ottenuti possono ritenersi soddisfacenti per quanto riguarda il confinamento delle regole definite all'interno del componente stesso, purtroppo però l'esito non è stato altrettanto apprezzabile per quanto riguarda l'isolamento del micro-frontend dalle influenze derivanti dal contesto di esecuzione. Per questo motivo si è deciso di adottare un incapsulamento basato su Shadow DOM, con l'obiettivo di ridurre al minimo i conflitti i pagina. In fase di dichiarazione dei futuri micro-frontend verrà quindi espressa in maniera esplicita la configurazione desiderata, questo al fine di applicare la gestione scelta, un esempio nel listato 5.6.

```
1 @Component({
2   selector: 'calcolatrice-tari',
3   templateUrl: './calcolatrice.component.html',
4   styleUrls: ['./calcolatrice.component.scss'],
5   encapsulation: ViewEncapsulation.ShadowDom
6 })
7 export class CalcolatriceComponent implements OnInit {
8     ...
9 }
```

Listato 5.6: Esempio di dichiarazione di un componente Angular incapsulato con Shadow DOM.

5.3.3 Unità di stile

All'interno della sezione 5.2.2 sono state riportate le considerazioni in merito al componente Calcolatrice TARI ed è stato analizzato il fenomeno riscontrato, ossia la mancanza di unità di stile tra i due micro-frontend in

esame, confermando le teorie riportate in sezione 3.5.1.
Tipicamente adottando un approccio a micro-frontend, la gestione dello stile non condiviso viene effettuata attraverso la definizione di regole specifiche per l'ambito del singolo componente in questione, come presentato in sezione 5.3.2, invece per realizzare una UX coesa e coerente vengono spesso condivise linee guida, componenti grafici ed elementi di stile attraverso la realizzazione di un design system [34].
Un principio chiave legato alle architetture a micro-frontend consiste nel mantenere i componenti il più disaccoppiati possibile, evitando di condividere codice e di instaurare forme di dipendenza, al fine di mantenere i team autonomi ed indipendenti. Questo concetto è stato ribadito più volte anche all'interno di questo scritto ma in materia di esperienza utente e design delle interfacce possono insorgere numerose complicazioni, come è stato sperimentato in questa situazione.
Adottare un sistema per condividere i blocchi elementari costitutivi delle interfaccia utente, è un processo che si scontra con il principio enunciato, il requisito di costruire interfacce dall'aspetto simile in grado di fornire un'esperienza utente consistente è però inderogabile. Pertanto in questo contesto ridurre il livello di disaccoppiamento può comunque portare a dei vantaggi tangibili.

Vista la necessità di seguire le direttive imposte da AgID, come dichiarato in sezione 2.3, il primo approccio è stato quello di valutare direttamente l'introduzione, all'interno del progetto, di uno degli strumenti offerti da Designers Italia, descritti in sezione 3.4.1.
Purtroppo però nessuna di queste librerie si è rivelata integrabile all'interno del sistema, Web Toolkit è stato infatti scartato in quanto deprecato, la sua evoluzione, ossia Boostrap Italia, è stata esclusa in quanto basata su jQuery e quindi difficilmente compatibile con il framework JavaScript scelto. Per lo stesso motivo non è stato selezionato il progetto React Kit mentre invece l'adozione della libreria Angular Kit, è stata ritenuta una strada non percorribile in quanto incompleta e all'apparenza non in manutenzione.

Una seconda ipotesi vagliata dal gruppo è stata quella di affidarsi ad altri progetti di questo tipo già presenti in azienda, per questo motivo sono stati svolti diversi incontri con i responsabili di vari team aziendali, in particolare sono stati presi i contatti con il team di infrastruttura di Sicr@web e con quello di Concilia.
I confronti avuti si sono rivelati estremamente utili e hanno offerto numerosi spunti interessanti, come preventivato però le soluzioni software adottate dai colleghi non rispondevano alle esigenze del progetto corrente, una condivisione delle soluzioni adottate da essi è stata pertanto esclusa come possibilità.
Il motivo risiede principalmente nel fatto che linee guida e stili adottati, fossero distanti da quanto cercato per il progetto e suggerito da AgID, inoltre questi software sono entrambi basati su una libreria di componenti grafici chiamata Kendo UI[6]. Siccome si tratta di una libreria disponibile solo sotto licenza, la soluzione è stata esclusa anche per via di dinamiche particolari legate alla partecipazione di bandi pubblici.

La decisione finale è stata dunque quella di sviluppare internamente la soluzione software, in modo da rispondere anche a quelle che sono le linee guida definite da AgID e lo stile grafico degli applicativi del gruppo Servizi al cittadino.
Per fare fronte all'esigenza si è quindi deciso di sviluppare una *pattern library*, contenente i componenti grafici di base per creare le interfacce dei micro-frontend del gruppo.
Il termine pattern library indica infatti un insieme di elementi base, come pulsanti e form di input, che gli sviluppatori possono utilizzare per realizzare pagine complesse, inoltre in aggiunta alle motivazioni ricercate permette di ottenere diversi vantaggi, come una maggiore velocità durante lo sviluppo ed una scalabilità superiore [14]. Gli elementi sviluppati all'interno di una pattern library devono essere atomici ed agnostici rispetto al dominio, seguendo quello che è il principio di singola responsabilità [57].

[6] `https://www.telerik.com/kendo-ui`

Il team di riferimento a cui verrà assegnata questa libreria è J-City Gov, ciò non significa che gli altri membri del gruppo non potranno contribuire ad essa ma, viste le esperienze passate, si è ritenuto preferibile affidare in consegna il progetto ad un unico team.
La motivazione risiede nel fatto che spesso approcci basati sulla libera condivisione portano ad una mancanza di visione di insieme e all'inconveniente che non vi siano figure di riferimento incaricate di preoccuparsi per le sorti del progetto.
Al fine di superare quelle che sono le difficoltà di comunicazione date dalla distanza tra le sedi e dalla suddivisione dell'organizzazione aziendale, oltre che per rendere più pratico ed immediato l'utilizzo dei componenti sviluppati all'interno della libreria, si è deciso di affidarsi ad uno strumento adottato anche dal team di Concilia che gli ha permesso di condividere la propria pattern library in maniera efficace ed efficiente con i membri dei gruppi aziendali Icaro e Sige. Il lavoro realizzato dai membri del team Concilia, mostrato in figura 5.6, si basa su una libreria chiamata Storybook[7].

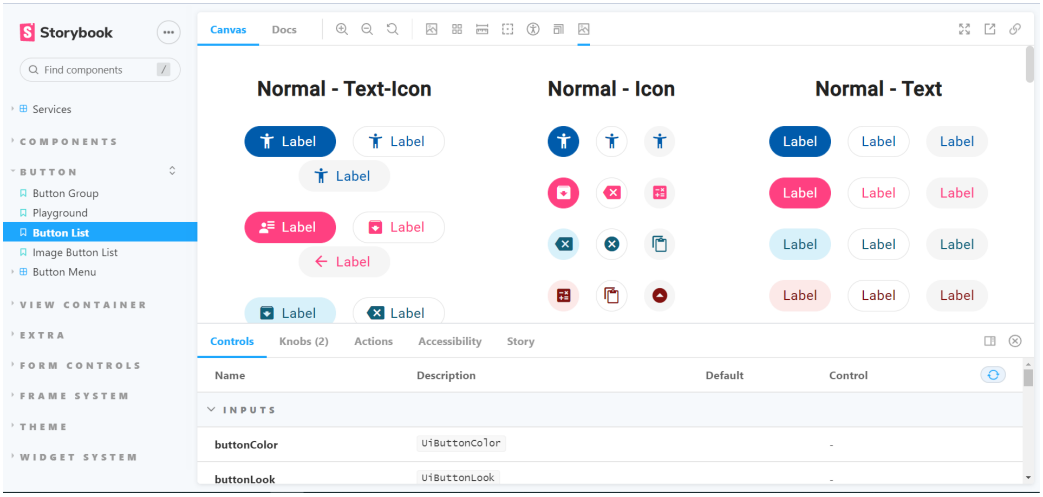

Figura 5.6: Esempio di Storybook del progetto aziendale Concilia.

Storybook è uno strumento open-source pensato per offrire un ambiente

[7]https://storybook.js.org/

dove poter gestire gli elementi che compongono l'interfaccia utente, agevola le fasi di sviluppo e test di questi, consentendo di operare in maniera isolata e permette di realizzare un catalogo all'interno del quale mostrare i contenuti realizzati, al quale è possibile associare la dovuta documentazione grazie alla definizione di quelle che chiama *story* [58].
Le story sono gli elementi fondanti del sistema e permettono di catturare e mostrare l'aspetto del componente in relazione ad un determinato stato, per descrivere interamente un componente vengono dunque scritte più storie, in modo da rappresentare tutti gli stati di interesse che l'elemento può assumere.
Lo strumento può essere inoltre esteso tramite un ampio ecosistema di componenti aggiuntivi che lo arricchiscono di varie funzionalità, come agevolazioni alle verifiche in materia di accessibilità o supporto allo sviluppo di elementi reattivi.

Conclusioni

In questa tesi sono state documentate le fasi principali che compongono il processo di trasformazione in corso all'interno di un reale contesto aziendale, il cui fine ultimo è quello di attuare in maniera ottimale l'integrazione e la migrazione di un insieme di applicativi caratterizzati da funzionalità per buona parte sovrapposte. Il tutto attraverso una evoluzione architetturale mirata a progettare un sistema basato su un approccio a microservizi e micro-frontend, in grado di ottimizzare tempo e risorse investite dall'organizzazione, consentendo a team full-stack di lavorare in piena autonomia e quindi di possedere la massima libertà di azione all'interno dei confini definiti dal dominio assegnato, a partire dal livello di persistenza sino alla gestione delle logiche di presentazione legate all'interfaccia utente. Grazie anche alla riprogettazione del modello di organizzazione del lavoro i team avranno dunque il pieno controllo del ciclo di sviluppo del software, dall'analisi, all'implementazione, fino alla distribuzione dei componenti appartenenti al proprio dominio di competenza.

Il caso di studio analizzato si è rivelato quindi molto efficace per mostrare come sia possibile approcciarsi ad una architettura a micro-frontend. Gli studi preliminari necessari, gli aspetti principali su cui basare la progettazione del sistema ed i punti essenziali su cui prestare maggiore attenzione, al fine di raggiungere con successo all'obiettivo prefissato, sono stati illustrati presentando prima lo stato dell'arte in materia e poi in forma pratica, grazie appunto all'aiuto di un caso di studio reale.
Il progetto aziendale è però ancora in corso, per questo motivo il lavoro pre-

sentato si è concentrato sull'analisi delle fasi iniziali che sono anche quelle più ricche di contenuti in materia dell'argomento principale di questa tesi, ossia delle architetture a micro-frontend. Nonostante ciò, lo scopo fondamentale del progetto, ossia la possibilità di evolvere il sistema attuale verso una nuova soluzione software, ottimizzando al massimo le risorse a disposizione e senza causare disservizi ai clienti attuali, sospendendo l'aggiornamento e l'evoluzione degli applicativi attualmente in uso, generando invece valore in maniera incrementale e costante rilasciando gradualmente parti del prodotto finale, è un vantaggio inequivocabile e già riscontrato durante lo sviluppo dei primi servizi discussi.

I lavori all'interno dell'organizzazione proseguiranno dunque come descritto all'interno dell'ultimo capitolo, ossia ultimando lo sviluppo della pattern library utile a garantire una maggiore unità di stile tra i micro-frontend sviluppati, attività necessaria al completamento e quindi al rilascio finale dei primi componenti, attualmente in fase di realizzazione e già ben avviati.
I passaggi successivi riguardano, come detto in fase di analisi, la rifattorizzazione dei servizi offerti da J-City Gov, allo scopo di conseguire risultati importanti in funzione dell'obiettivo di cessare l'applicativo già entro la fine dell'anno corrente.
Durante la realizzazione di questi sviluppi, verranno collezionate tutte le informazioni utili al prosieguo delle indagini funzionali al conseguimento delle decisioni finali, riguardanti la definizione dell'applicazione container definitivo e quindi al passaggio verso l'ultima fase del progetto in questione. Inoltre, verrà promossa una cultura dell'innovazione e della condivisione, in questo periodo infatti continueranno in maniera intensiva le attività di evoluzione per quanto concerne il ciclo di rilascio, in linea con le direttive aziendali, e soprattutto sarà stimolato uno scambio di informazioni continuo, attività fondamentale vista la nuova forma di organizzazione del lavoro, al fine di promuovere la condivisione delle conoscenze acquisite dai team viste le minori occasioni di confronto, per via della maggiore indipendenza conquistata.

Bibliografia

[1] Technology radar: Micro frontends, 2016. https://www.thoughtworks.com/radar/techniques/micro-frontends.

[2] Alberto Marinelli. Il web: Un meraviglioso giardino... aperto o chiuso? *CASPUR Annual Report*, 2012.

[3] Raoni Kulesza, Marcelo Fernandes de Sousa, Matheus Lima Moura de Araújo, Claudiomar Pereira de Araújo, and Aguinaldo Macedo Filho. *Evolution of Web Systems Architectures: A Roadmap*, pages 3–21. Springer International Publishing, Cham, 2020. ISBN 978-3-030-35102-1. doi: 10.1007/978-3-030-35102-1_1. https://doi.org/10.1007/978-3-030-35102-1_1.

[4] Anna Montelius. An exploratory study of micro frontends. Master's thesis, Linköping University, 2021.

[5] Jr Emmit A. Scott. *SPA Design and Architecture Understanding Single-Page Web Applications*. Manning Publications, Shelter Island, first edition, 2016.

[6] Severi Peltonen, Luca Mezzalira, and Davide Taibi. Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 136:1–16, 2021. ISSN 0950-5849. doi: \url{https://doi.org/10.1016/j.infsof.2021.106571}. https://www.sciencedirect.com/science/article/pii/S0950584921000549.

[7] Michael Geers. Micro frontends: extending the microservice idea to frontend development, 2021. https://micro-frontends.org.

[8] Luis Florez, Eduardo Rosa, Duy V Nguyen, and Sandro De Santis. *Evolve the Monolith to Microservices with Java and Node*. IBM Redbooks, 2016.

[9] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen. Challenges when moving from monolith to microservice architecture. In Irene Garrigós and Manuel Wimmer, editors, *Current Trends in Web Engineering*, pages 32–47, Cham, 2018. Springer International Publishing. ISBN 978-3-319-74433-9.

[10] James Lewis and Martin Fowler. Microservices, 2014. https://martinfowler.com/articles/microservices.html.

[11] Sam Newman. *Building Microservices*. O'Reilly Media, 2015.

[12] Luca Mezzalira. *Building Micro-Frontends*. O'Reilly Media, first edition, November 29, 2021.

[13] Cam Jackson. Micro frontends, 2019. https://martinfowler.com/articles/micro-frontends.html.

[14] Michael Geers. *Micro Frontends in Action*. Manning Pubns Co, 2020.

[15] Paul Brook. Microservice front-end – a modern approach to the division of the front, 2018. https://www.smartspate.com/microservice-front-end/.

[16] Andrey Pavlenko, Nursultan Askarbekuly, Swati Megha, and Manuel Mazzara. Micro-frontends: application of microservices to web frontends. *Journal of Internet Services and Information Security (JISIS)*, 10 (2):49–66, May 2020. doi: 10.22667/JISIS.2020.05.31.049.

[17] World Wide Web Consortium. Web components specifications, 2019. https://github.com/WICG/webcomponents.

[18] Sam Newman. Pattern: Backends for frontends, 2015. https://samnewman.io/patterns/architectural/bff/.

[19] Gruppo Maggioli. Sito ufficiale, 2021. https://www.maggioli.it.

[20] Ilaria Vesentini. Maggioli, terza acquisizione in spagna nel giro di tre anni. *Il Sole 24 Ore*, page 14, 6 2019.

[21] Maggioli Tributi, 2021. https://www.maggiolitributi.it.

[22] Il gruppo maggioli oltre la pandemia: chiude il 2020 con un bilancio positivo. *Corriere Romagna*, page 25, 4 2021.

[23] Liferay Sito Ufficiale, 2021. https://www.liferay.com/.

[24] Filipowicz Piotr and Ziółkowska Katarzyna. *Liferay 6.x Portal Enterprise Intranets Cookbook*. Packt Publishing Limited, 2015.

[25] Liferay Help Center, 2021. https://help.liferay.com.

[26] Navin Agarwal. *Liferay Portal 6.2 Enterprise Intranets*. Packt Publishing Limited, 2015.

[27] Municpium App Sito Ufficiale, 2021. https://www.municipiumapp.it/web/.

[28] Drupal Association. Drupal project, 2021. https://www.drupal.org.

[29] Martin Fowler. Stranglerfigapplication, 2004. https://martinfowler.com/bliki/StranglerFigApplication.html.

[30] Sam Newman. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, 2019.

[31] Kyle Brown. Apply the strangler fig application pattern to microservices applications, 2017. https://developer.ibm.com/articles/cl-strangler-application-pattern-microservices-apps-trs/.

[32] Ed Price, Alex Buck, and Neil Peterson. Strangler fig pattern, 2021. https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler-fig.

[33] Manfred Steyer. Multi-framework and -version micro frontends with module federation: The good, the bad, the ugly, 2021. https://www.angulararchitects.io/en/aktuelles/multi-framework-and-version-micro-frontends-with-module-federation-the-good-the-bad-the-ugly/.

[34] Kevin C. Y. Chen. Micro frontend guide: Technical integrations, 2021. https://www.trendmicro.com/it_it/devops/21/h/micro-frontend-guide-technical-integrations.html.

[35] J. Fernando Sánchez-Rada, Alberto Pascual, Enrique Conde, and Carlos A. Iglesias. A big linked data toolkit for social media analysis and visualization based on w3c web components. In Hervé Panetto, Christophe Debruyne, Henderik A. Proper, Claudio Agostino Ardagna, Dumitru Roman, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, pages 498–515, Cham, 2018. Springer International Publishing. ISBN 978-3-030-02671-4.

[36] Caleb Williams. An introduction to web components, 2019. https://css-tricks.com/an-introduction-to-web-components/.

[37] Manish Virmani. Understanding devops & bridging the gap from continuous integration to continuous delivery. In *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, pages 78–82, 2015. doi: 10.1109/INTECH.2015.7173368.

[38] Manfred Steyer. Multi-framework and -version micro frontends with module federation: The good, the bad, the ugly, 2021. https://www.angulararchitects.io/en/aktuelles/6-steps-to-your-angular-based-microfrontend-shell/.

[39] Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward Smith, Collin Winter, and Emerson Murphy-Hill. Advantages and disadvantages of a monolithic repository: A case study at google. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 225–234, 05 2018. doi: 10.1145/3183519.3183550.

[40] Vincent Driessen. A successful git branching model, 2010. `https://nvie.com/posts/a-successful-git-branching-model/`.

[41] Martin Fowler. Continuous integration, 2006. `https://www.martinfowler.com/articles/continuousIntegration.html`.

[42] G Ann Campbell and Patroklos P Papapetrou. *SonarQube in action*. Manning Publications Co., 2013.

[43] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017. doi: 10.1109/ACCESS.2017.2685629.

[44] Emiliano Casalicchio. *Container Orchestration: A Survey*, pages 221–235. Springer International Publishing, Cham, 2019. ISBN 978-3-319-92378-9. doi: 10.1007/978-3-319-92378-9_14. URL `https://doi.org/10.1007/978-3-319-92378-9_14`.

[45] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014. doi: 10.1109/MCC.2014.51.

[46] Burns Brendan, Beda Joe, and Hightower Kelsey. *Kubernetes: Up & Running. Dive into the Future of Infrastructure*. O'Reilly Media, second edition, 2019.

[47] What is kubernetes?, 2021. `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/`.

[48] Josef Spillner. Quality assessment and improvement of helm charts for kubernetes-based cloud applications. *CoRR*, abs/1901.00644, 2019.

[49] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Dimensions of devops. In Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 212–217, Cham, 2015. Springer International Publishing. ISBN 978-3-319-18612-2.

[50] Yujuan Jiang and Bram Adams. Co-evolution of infrastructure and source code - an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 45–55, 2015. doi: 10.1109/MSR.2015.12.

[51] Navin Sabharwal, Sarvesh Pandey, and Piyush Pandey. *Infrastructure-as-Code Automation Using Terraform, Packer, Vault, Nomad and Consul*. Apress, 2021.

[52] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing Idempotence for Infrastructure as Code. In David Hutchison, Takeo Kanade, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, David Eyers, Karsten Schwan, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, and Bernhard Steffen, editors, *14th International Middleware Conference (Middleware)*, volume LNCS-8275 of *Middleware 2013*, pages 368–388, Beijing, China, December 2013. Springer. doi: 10.1007/978-3-642-45065-5_19. Part 4: Services.

[53] Terraform. Sito ufficiale, 2021. `https://www.terraform.io/`.

[54] Andreas Steffens, Horst Lichter, and Jan Simon Döring. Designing a next-generation continuous software delivery system: Concepts and architecture. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 1–7, 2018.

[55] Stojan Anastasov. Setting up gitlab ci/cd for android projects, 2018. `https://about.gitlab.com/blog/2018/02/14/setting-up-gitlab-ci-for-android-projects/`.

[56] GitLab. documentazione ufficiale, 2021. `https://docs.gitlab.com/ee/ci/pipelines/`.

[57] Robert C Martin, James Newkirk, and Robert S Koss. *Agile software development: principles, patterns, and practices*, volume 2. Prentice Hall Upper Saddle River, NJ, 2003.

[58] Storybook. Sito ufficiale, 2021. `https://storybook.js.org/`.