Politecnico di Torino

Politecnico di Torino

L.M In Ingegneria Informatica (A.I and Data Analytics)
A.a. 2023/2024
Sessione di Laurea Ottobre 2024

Realizzazione stream di dati e applicazione Web per l'analisi di movimentazioni contabili

Relatori:
Prof. Masala Enrico
Co-Relatori:
Dott.ssa. Cucillo Valentina
Ing. Desimoni Federico

Candidati:
Morleo Ethan

Al mio amore Virginia,
Alla mia famiglia

ABSTRACT

La presente tesi, realizzata in un contesto aziendale, esplora lo sviluppo di un'applicazione web di gestione e analisi di dati in ambito bancario.

Gli elementi focali sono gli strumenti e framework tecnologici specifici per ottimizzare e migliorare l'esperienza utente nell'ambito dell'analisi delle movimentazioni contabili.

Il lavoro si concentra sulla progettazione e implementazione di precise fasi di estrazione, trasformazione e caricamento (ETL) dei dati e sulla realizzazione di una piattaforma web che fornisce i dati rielaborandoli ulteriormente secondo le necessità degli utenti garantendo scalabilitá e usabilitá.

In particolare, si descriveranno le scelte architetturali e tecnologiche riguardanti: il flusso dei dati e interazione con i database, i framework per la realizzazione di servizi di backend robusti e scalabili, la realizzazione di un'interfaccia grafica moderna, per garantire un'interfaccia utente intuitiva e reattiva. La tesi illustra anche le sfide affrontate e le soluzioni adottate durante il ciclo di sviluppo, inclusa la gestione delle performance e l'integrazione con sistemi legacy.

I risultati ottenuti dimostrano una significativa ottimizzazione dei processi e un miglioramento dell'interazione con gli utenti finali. Questo studio spinge sull'innovazione e transizione digitale del settore bancario, offrendo spunti pratici per future implementazioni e ricerche.

Indice

Capitolo 1

Dominio e ambiti di sviluppo

1.1 Introduzione al dominio

In questi anni, in cui i mercati globali vivono una crescente complessità e le esigenze aziendali sono in continua evoluzione, l'analisi avanzata dei dati è diventata sempre più importante e rappresenta un elemento cruciale per il successo e la competitività delle imprese. In questo scenario, Target Reply si pone come una realtà di primo piano nel panorama delle soluzioni di Analytics, Big Data e Data Science, offrendo un supporto end-to-end alle aziende lungo l'intero processo di gestione e valorizzazione del dato. Target Reply, società del gruppo Reply, è specializzata nella progettazione e realizzazione di soluzioni innovative per l'integrazione dei dati, il data modelling, l'analisi predittiva e prescrittiva. L'azienda si distingue per l'adozione di strumenti e approcci all'avanguardia, in ambiti come il Fast Data e il Machine Learning, settori in cui si utilizzano nuove tecnologie emergenti.

Questo lavoro di tesi, realizzato in collaborazione con Target Reply, in cui svolgo il ruolo di data engineer, si basa su una serie di componenti e fasi lavorative che hanno permesso la realizzazione di uno strumento di analisi delle movimentazioni contabili, richieste da un importante cliente nel settore bancario, focalizzandosi sulle sfide riguardanti: il processo di analisi ed elaborazione dei dati e lo sviluppo e gestione di un'applicazione web realizzata per gli utenti finali, cui compito è l'ulteriore analisi dei dati proposti.

Figura 1.1: Logo target reply

1.1.1 Movimentazioni contabili

Il settore bancario è uno dei più critici e regolamentato tra i mercati globali. Le movimentazioni contabili sono le registrazioni di tutte le transazioni economiche, accompagnate da scritture contabili, che devono rispettare alcuni formalismi, assicurando che ogni transazione sia debitamente riportata sia come credito che come debito nei conti corrispondenti.

La gestione delle movimentazioni contabili è il fulcro delle operazioni finanziarie, la loro comprensione è essenziale per una corretta gestione della contabilità e per garantire la trasparenza, affidabilità e accuratezza delle informazioni finanziarie su cui si basano alcune decisioni aziendali.

Le movimentazioni contabili sono rilevanti per una serie di aspetti tra cui:

1. Una visione chiara e trasparente delle risorse disponibili e degli obblighi finanziari.

2. Le decisioni sugli investimenti e finanziamenti che si basano su report finanziari, e previsioni generati dai dati delle movimentazioni contabili.

3. Rispetto di normative contabili e fiscali. Le movimentazioni contabili sono la base su cui si fondano una serie di adempimenti legali tra cui i bilanci annuali e le dichiarazioni fiscali.

4. Una visione dettagliata delle spese e dei ricavi, permettendo una gestione finanziaria più efficiente.

In conclusione, la capacità di analizzare efficacemente i dati delle movimentazioni contabili è fondamentale per le istituzioni bancarie e l'analisi di questi dati permette non solo di monitorare la salute finanziaria, ma anche di analizzare dei comportamenti anomali, gestire il rischio, ottimizzare i processi decisionali e offrire servizi personalizzati ai clienti.

Questo progetto, nasce dalla crescente complessità delle operazioni bancarie e il volume sempre maggiore di dati che richiedono strumenti avanzati e specifici per l'analisi e la gestione di queste informazioni.

1.2 Contributo personale al progetto e ambiti di sviluppo

Il mio ruolo, come consulente per il cliente, si basa sulla realizzazione di soluzioni alle sfide tecnologiche e implementative richieste per quanto concerne i seguenti ambiti:

- La gestione di particolari tipi di crediti, organizzandoli in modo che ognuno sia identificato, migliorando la tracciabilità tra i diversi attori coinvolti. La gestione dei crediti prevede che, una volta accettati, essi non possano essere annullati, e permette l'applicazione di diverse modalità operative, in base alla tipologia del credito. Inoltre, è previsto un processo di ricessione dei crediti a terzi, con una gestione dettagliata delle informazioni contrattuali e la suddivisione dei crediti in diversi portafogli, secondo regole specifiche di gestione.

- La gestione di tutto il ciclo di vita di fondi utilizzabili in caso di alti rischi, con la definizione di una serie di regole per la revisione e l'attualizzazione temporale degli stessi. Sarà svolta particolare attenzione sulle scritture contabili di tali fondi che dovranno seguire regole ben precise.

- La gestione di crediti deteriorati,che anche in questo caso, avrà un importante focus sui processi di estrazione dei movimenti contabili provenienti da legacy per la rielaborazione ed esecuzione di ulteriori scritture contabili.

L'obiettivo finale è fornire all'istituzione bancaria uno strumento tecnologico che sia in grado di aggregare, elaborare e visualizzare i dati in modo efficiente e intuitivo in modo tale da supportare gli utenti finali utilizzatori dell'app nel processo decisionale e velocizzando la loro attività, mediante funzionalità di analisi approfondita, reportistica personalizzata e monitoraggio in tempo reale.

Il fulcro del lavoro risiede nelle tecnologie utilizzate per l'estrazione e trasformazione del dato messo poi a disposizione in una web application che sfrutta una combinazione di tecnologie per quanto riguarda sia il Backend che il Frontend, messe a disposizione dall'architettura di sviluppo del cliente.

L'applicazione mira a semplificare il processo di analisi, riducendo il tempo e lo sforzo necessario per estrarre informazioni significative dalle movimentazioni contabili.

Il progetto risponde a una necessità concreta del cliente: l'esigenza di uno strumento efficace e sicuro per l'analisi dei dati contabili

Capitolo 2

Architetture e tecnologie utilizzate

2.1 Architetture Software

L'architettura software rappresenta la struttura fondamentale su cui si basa qualsiasi applicazione complessa, in cui si definisce non solo la disposizione dei componenti software, ma anche le modalità con cui questi interagiscono tra loro e con l'ambiente circostante.

In questo progetto bisogna garantire: affidabilità, sicurezza e scalabilità, per questo la scelta dell'architettura software adeguata risulta cruciale per il successo operativo.

In questo capitolo si esplorano le principali architetture software che verranno utilizzate per la realizzazione del progetto. In particoloare : i sistemi a microservizi, che offrono flessibilità e scalabilità; i sistemi batch, essenziali per l'elaborazione massiva di dati e per la gestione di operazioni periodiche e i sistemi event-driven basati cioè sulla ricezione di eventi derivanti da diverse sorgenti per l'elaborazione real time dei dati.

Verranno descritti i vantaggi e gli svantaggi di ogni architettura a seguito di una descrizione dettagliata, in modo da fornire delle considerazioni tecniche sulla adozione nel nostro dominio di interesse. Queste analisi permetteranno di comprendere meglio le implementazioni delle architetture software per soddisfare le esigenze degli ambiti di sviluppo del progetto.

2.1.1 Architettura basata su microservizi

L'architettura a microservizi [2] rappresenta un approccio per la realizzazione di un'applicazione in cui questa è suddivisa in una serie di piccoli servizi indipendenti, ognuno dei quali gestisce una specifica funzionalità del sistema. Questi servizi comunicano tra loro tramite API, generalmente usando protocolli leggeri.

Ogni microservizio è sviluppato, distribuito e gestito in modo indipendente, consentendo una maggiore flessibilità rispetto alle architetture monolitiche tradizionali.

I microservizi sono utilizzati per costruire applicazioni modulari che possono essere facilmente aggiornate, scalate e mantenute e permettendo una maggiore agilità nello sviluppo e nella gestione separata di molteplici funzioni.

L'architettura a microservizi si basa su alcuni principi chiave:

- **Decomposizione in Servizi**: L'applicazione è scomposta in piccoli servizi indipendenti, ognuno dei quali è responsabile di una specifica funzione. Questo consente di aggiornare e scalare i servizi in modo indipendente.
- **API e Comunicazione**:I microservizi comunicano tra loro attraverso API, la comunicazione può avvenire tramite protocolli sincroni (come HTTP/REST) o asincroni (come messaggi tramite broker come Kafka).
- **Contenitori e Orchestrazione**: I microservizi sono spesso distribuiti utilizzando container Docker, che forniscono un ambiente isolato per ogni servizio. L'orchestrazione dei container, gestita da strumenti come Kubernetes, consente di automatizzare la distribuzione, il bilanciamento del carico, la scalabilità e il recupero in caso di guasti.
- **Persistenza Decentralizzata**: Ogni microservizio può gestire la propria base dati, permettendo una maggiore autonomia e un utilizzo diversificato di tecnologie di storage in base alle esigenze specifiche del servizio.

2.1.1.1 Vantaggi e svantaggi architettura a microservizi

L'adozione dell'architettura a microservizi è caratterizzata da numerosi vantaggi, i principali sono:

- **Flessibilità nello sviluppo**: L'indipendenza dei servizi, facilita il team di sviluppo che può lavorare contemporaneamente su più componenti diversi, riducendo così i tempi di sviluppo.
- **Scalabilità**: I microservizi possono anche essere scalati in modo indipendente, permettendo di allocare dinamicamente le risorse solo per i servizi che ne hanno bisogno. Questo approccio è più efficiente rispetto alla scalabilità verticale di altre architetture, in quanto riduce lo spreco di risorse, i costi nell'acquisto di nuovo hardware e aumenta la flessibilità generale.
- **Resilienza**: L'indipendenza dei microservizi non generalizza un problema in un servizio su tutto il sistema. Le anomalie possono essere isolate e gestite senza influenzare altre parti dell'applicazione, aumentando la resilienza complessiva del sistema.
- **Manutenibilità**: La separazione dei servizi in unità più piccole e gestibili facilita anche la manutenzione e l'aggiornamento del codice sorgente. Le modifiche a un microservizio possono essere implementate, testate e distribuite senza impattare il resto dell'applicazione, riducendo i rischi di regressione.
- **Integrazione con diverse tecnologie**: L'architettura a microservizi si presta all'integrazione con nuove tecnologie, come il cloud computing e i container, rendendo più facile adottare soluzioni innovative e mantenere un adeguato aggiornamento tecnologico continuo.

I principali svantaggi, nell'adozione di una architettura a microservizi, sono i seguenti:

- **Gestione e orchestrazione complesse**: L'aumento del numero di servizi comporta una maggiore complessità nella gestione dell'infrastruttura.E‘ necessario implementare strumenti avanzati per l'orchestrazione, il monitoraggio e il logging.
- **Problemi di Rete**: La comunicazione tra microservizi avviene tramite rete, il che introduce potenziali latenze. La gestione di questi problemi richiede un'attenta progettazione delle API e l'implementazione di meccanismi di resilienza.

- **Overhead di Sviluppo**: L'implementazione di un'architettura a microservizi richiede un investimento significativo nella definizione di API, nella gestione delle versioni e nell'automazione delle pipeline di CI/CD. Questo overhead può inizialmente rallentare lo sviluppo e richiede competenze nelle fasi di sviluppo e distribuzione avanzate e specialistiche.

- **Coerenza dei dati**: I dati sono distribuiti tra piu' microservizi, garantire la coerenza degli stessi diventa una sfida. Si rende necessario implementare strategie di gestione delle transazioni distribuite, che possono complicare l'architettura complessiva.

2.1.1.2 Confronto con architettura legacy

I sistemi mainframe [1], al contrario dei microservizi, sono sistemi di elaborazione centralizzati, progettati per gestire grandi volumi di transazioni e dati che rimangono ancora oggi una componente fondamentale per alcune infrastrutture IT.

Questi sistemi garantiscono un'alta disponibilità e affidabilità, con meccanismi integrati per garantire che i servizi siano sempre operativi, anche in caso di guasti hardware o software.

I principali componenti di un sistema mainframe sono:

- CPU Multiprocessore, altamente ottimizzati per l'elaborazione parallela.

- Memoria RAM ad alta Capacità con accesso a bassa latenza, per supportare l'elaborazione veloce dei dati.

- Storage Centralizzato, spesso basati su dischi rigidi ad alte prestazioni o su tecnologie SSD con elevate capacità di memorizzazione e accesso rapido ai dati.

- Sistemi Operativi Specializzati come z/OS, ottimizzati per l'elaborazione di grandi volumi di dati e transazioni. Tali sistemi operativi includono funzionalità avanzate di gestione delle risorse, sicurezza e virtualizzazione.

Nonostante l'architettura basata su mainframe possa risultare obsoleta, essa continua a garantire vantaggi significativi, soprattutto in termini di prestazioni elevate e scalabilità verticale.

Tuttavia, l'architettura mainframe presenta anche notevoli complicanze tra cui: i costi di manutenzione e gestione elevati, sia per quanto riguarda l'hardware che il personale specializzato necessario per operare e manutenere questi sistemi.

L'architettura mainframe si basa su un modello monolitico e centralizzato, rendendo meno flessibile la gestione del software: ogni modifica richiede di intervenire su una grande parte del codice sorgente o dell'infrastruttura, aumentando i tempi di rilascio e manutenzione. Scalare un sistema mainframe richiede spesso l'acquisto di costosi hardware, piuttosto che la scalabilità distribuita tipica dei microservizi.

In termini di gestione operativa, l'architettura a microservizi si adatta meglio a infrastrutture cloud-native, dove l'automazione e l'orchestrazione dei servizi (ad esempio, con Kubernetes) consentono una gestione dinamica delle risorse. Al contrario, i mainframe sono sistemi proprietari, che richiedono competenze specifiche e infrastrutture hardware dedicate, con costi di mantenimento elevati e tempi di aggiornamento più lunghi.

In conclusione, l'architettura a microservizi offre flessibilità, modularità e scalabilità e un ciclo di sviluppo rapido e un'alta resilienza, invece l'architettura mainframe è orientata a garantire potenza, affidabilità e sicurezza in ambienti che richiedono gestione di transazioni ad alto volume, ma con una minore flessibilità e scalabilità rispetto ai microservizi.

2.1.1.3 Evoluzione e adozione futura dell'architettura a microservizi

L'adozione di architetture a microservizi è particolarmente diffusa nelle organizzazioni che cercano di modernizzare le loro infrastrutture IT, aumentando l'agilità e la capacità di innovazione.

In futuro, ci si aspetta che questa architettura continui a evolversi, con un'enfasi crescente sull'uso di tecnologie di orchestrazione avanzata, come il serverless computing e il service mesh (ad esempio, Istio). Inoltre, l'integrazione con tecnologie di intelligenza artificiale e machine learning, per migliorare la personalizzazione e l'automazione dei servizi, rappresenta una direzione chiave per l'adozione dei microservizi.

2.1.2 Architettura event-driven

In questa sezione si analizza un'architettura che ha come scopo l'elaborazione di eventi cioè cambiamenti o azioni del sistema, in tempo reale [4]. Questa tecnica, permette

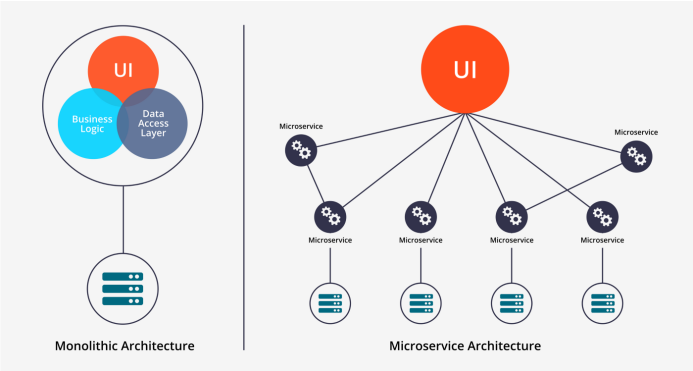

Figura 2.1: Architettura monolitica e a microservizi a confronto

di elaborare e analizzare i dati al momento della loro generazione, consentendo alle applicazioni di ricevere immediatamente nuove informazioni, tale architettura riscuote sempre più consensi in quanto la rapidità nell'elaborazione e nella reazione ai dati può offrire un significativo vantaggio competitivo. I principali componenti per realizzare tale architettura sono i seguenti:

- Sorgenti del dato cioè le fonti che generano i dati in modo continuo.
- Un broker di messaggi, come Apache Kafka utilizzato per raccogliere e distribuire i dati in streaming dalle sorgenti ai consumatori, il quale gestisce il buffering, la coda e la distribuzione dei dati, garantendo che ogni messaggio venga consegnato ai consumatori appropriati. In questo progetto verrà fatto uso di Kafka, in quanto altamente scalabile e supporta il trattamento di milioni di messaggi al secondo, rendendolo ideale per architetture di streaming real-time.
- Motore di elaborazione, in grado di elaborare i dati in tempo reale, applicando logiche di trasformazione, aggregazione, filtraggio e analisi dei dati

2.1.2.1 Vantaggi e svantaggi sistemi real-time

I vantaggi di una architettura real time sono i seguenti:

- **Reattività**: Lo streaming real-time permette alle applicazioni di reagire immediatamente ai nuovi dati, offrendo diversi vantaggi come: il rilevamento tempestivo di anomalie e l'ottimizzazione sofisticata in tempo reale di alcuni componenti.

- **Elaborazione continua**: A differenza dei sistemi batch, dove i dati vengono elaborati in blocchi, lo streaming real-time permette l'elaborazione continua dei dati. Questo significa che i dati sono sempre aggiornati e le decisioni possono essere prese sulla base delle informazioni più recenti.

- **Flessibilità** : Le architetture di streaming real-time sono altamente flessibili e possono essere adattate per gestire una varietà di scenari di utilizzo, da semplici pipeline di dati a complesse applicazioni di intelligenza artificiale in tempo reale.

- Riduzione tempi di latenza: Lo streaming real-time minimizza la latenza tra la generazione dei dati e la loro analisi, consentendo una visibilità quasi istantanea sulle operazioni aziendali e migliorando la capacità di rispondere rapidamente a cambiamenti critici.

L'utilizzo di questa architettura comporta anche degli svantaggi:

- **Complessità implementazione**: l'implementazione di un'architettura di streaming real-time è complessa e richiede competenze avanzate in diverse tecnologie. La configurazione e la gestione di componenti come Kafka sono sfidanti e richiedono una gestione e un monitoraggio continuo.

- **Gestione dello stato e delle transazioni**: gestire lo stato in un sistema di streaming real-time è più complicato rispetto a un sistema batch. Garantire la consistenza dei dati, specialmente in caso di guasti o interruzioni, richiede un'attenta progettazione e l'uso di strumenti avanzati per il coordinamento e il ripristino.

- **Difficoltà debugging e monitoraggio**: il debugging e il monitoraggio di applicazioni di streaming in tempo reale sono più complessi rispetto ai sistemi tradizionali. La natura continua e distribuita dei dati rende difficile tracciare e risolvere i problemi, richiedendo strumenti specializzati e una forte competenza tecnica.

- **Latenza di propagazione**: anche se la latenza è generalmente bassa, ci sono scenari in cui la latenza di propagazione dei dati può aumentare, specialmente in architetture distribuite complesse. Questo può influenzare la tempestività delle decisioni basate sui dati.

2.1.2.2 Evoluzione e adozione futura dei sistemi real-time

L'architettura di streaming real-time è destinata a evolversi in modo significativo nei prossimi anni, visto l'aumento esponenziale del volume di dati generati e dalla crescente domanda di analisi in tempo reale.

In futuro, questa architettura vedrà un'integrazione più stretta con l'intelligenza artificiale e il machine learning, consentendo non solo di reagire ai dati in tempo reale, ma anche di prevedere eventi futuri. Inoltre, con la nuova tecnologia del 5G e l'uso sempre più massivo di dispositivi IoT, le architetture di streaming real-time verranno utilizzate maggiormente, gestendo flussi di dati provenienti da miliardi di dispositivi con latenza ultra bassa.

L'uso di infrastrutture basate sul cloud e l'orchestrazione di microservizi distribuiti, aumenteranno la scalabilità e la resilienza, consentendo una gestione dinamica e automatizzata delle risorse.

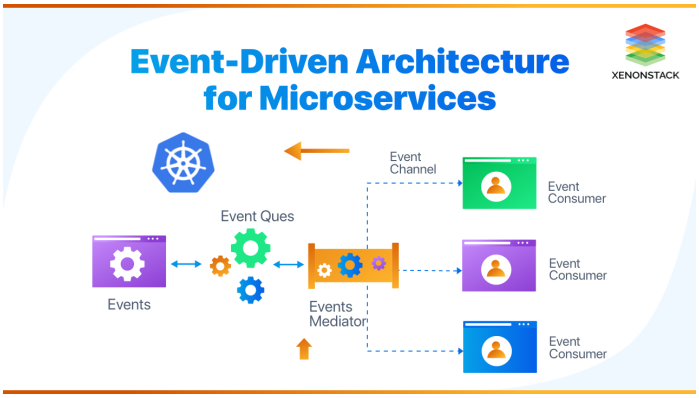

Figura 2.2: Architettura event-driven

2.1.3 Batch processing architecture

I sistemi batch [3] sono utilizzati per l'elaborazione di grandi volumi di dati in modo automatico e non interattivo, eseguendo una serie di operazioni predefinite su un insieme di dati in un tempo specifico.

L'elaborazione batch è utilizzata per garantire che tutte le operazioni siano completate correttamente entro un certo periodo. Nonostante l'evoluzione verso architetture più moderne, i sistemi batch rimangono una componente critica ad esempio per le operazioni in ambito bancario.

L'architettura di un sistema batch è progettata per eseguire lavori di elaborazione intensiva su grandi volumi di dati senza l'intervento umano, i componenti chiave per la realizzazione dell'architettura includono:

- **Scheduler di Job**: Il cuore di un sistema batch è lo scheduler, ha il compito di coordinare l'esecuzione dei job batch secondo una pianificazione predefinita. In questo progetto verrà approfondito l'utilizzo di IBM Tivoli Workload Scheduler come orchestratore di job.
- **Pipeline di Elaborazione**: I dati vengono elaborati in una sequenza di passaggi, spesso chiamata pipeline. Ogni fase può includere operazioni di estrazione, trasformazione, aggregazione e caricamento (ETL) dei dati. Queste pipeline sono progettate per essere robuste e tolleranti ai guasti.
- **Persistenza e Storage**: I sistemi batch lavorano tipicamente su grandi dataset che risiedono in database relazionali o data warehouse. È comune utilizzare tecnologie come Oracle, IBM DB2 per gestire i dati.

2.1.3.1 Vantaggi e svantaggi dei sistemi batch

I vantaggi nell'utilizzo di sistemi batch sono:

- Efficienza nell'elaborazione di grandi volumi di dati in quanto i sistemi batch sono ideali per attività che richiedono un alto livello di elaborazione.
- Automatizzazione e affidabilità, infatti dopo una fase di configurazione iniziale, un sistema batch può eseguire operazioni complesse in modo completamente automatizzato, riducendo la necessità di intervento manuale e minimizzando gli errori umani.
- Utilizzo ottimizzato delle risorse: poichè l'elaborazione batch avviene tipicamente durante periodi di basso utilizzo del sistema (come la notte), consente di ottimizzare l'uso delle risorse IT, sfruttando la capacità del sistema che altrimenti rimarrebbe inutilizzata.

Gli svantaggi di questa architettura sono invece i seguenti:

- Assenza di interattività, i job batch infatti, vengono eseguiti secondo una programmazione predeterminata, non sono adatti per operazioni che richiedono una risposta immediata o un'interazione continua con l'utente. Questo li rende inadatti per applicazioni in tempo reale.

- Latenza temporale, i batch introducono una latenza intrinseca, poichè i dati devono aspettare il momento programmato per essere elaborati. In un contesto bancario, ciò può significare che alcune operazioni non siano aggiornate in tempo reale.

- Rischio di errori accumulati, la presenza di un errore in un job batch può influenzare l'intero processo, accumulando errori che potrebbero non essere rilevati fino alla fine del ciclo di elaborazione. Questo può richiedere il riavvio del processo causando ritardi significativi e potenziali perdite di dati.

- Dipendenza da sistemi legacy: Molti sistemi batch sono basati su tecnologie legacy che possono essere difficili da integrare con nuove architetture e tecnologie moderne. Questo limita la flessibilità e può creare ostacoli nell'implementazione di nuove soluzioni IT.

2.1.3.2 Evoluzione e adozione futura dei sistemi batch

Nel futuro, i sistemi batch si potranno evolvere rispetto alle crescenti esigenze di elaborazione in tempo reale e all'integrazione con nuove tecnologie come il cloud computing e il machine learning. Alcuni dei trend futuri includono:

- **Integrazione con Big Data e analisi in tempo reale**: Con l'aumento dei volumi di dati, si cercherà di combinare l'elaborazione batch con tecnologie di big data per migliorare l'analisi e la predittività. Questo può includere l'uso di tecnologie come Apache Spark o Flink per elaborare grandi dataset in tempo reale, riducendo la dipendenza dai batch tradizionali.

- **Cloud**: migrazione dei processi batch verso ambienti cloud o ibridi, sfruttando la scalabilità e la flessibilità del cloud computing. Questo consente di eseguire batch job su risorse cloud durante i periodi di picco, riducendo i costi e migliorando l'efficienza.

- Automazione avanzata e RPA: L'integrazione di soluzioni di automazione avanzata e robot process automation (RPA) nei sistemi batch permetterà l'ulteriore ottimizzazione delle operazioni di back-office, migliorando l'efficienza e riducendo il rischio di errori umani.

- **Adozione di architetture Event-Driven**: Un altro trend emergente è l'adozione di architetture event-driven che combinano l'elaborazione batch con l'elaborazione in tempo reale. Questo permette di rispondere più rapidamente agli eventi, riducendo la latenza e migliorando l'efficienza operativa.

In conclusione, l'adozione di nuove tecnologie e architetture porterà a una graduale trasformazione dei sistemi batch, rendendoli più flessibili e adatti alle esigenze moderne per quanto concerne l'elaborazione dei dati.

2.2 Tecnologie utilizzate

All'interno del contesto aziendale del cliente, lo sviluppo delle soluzioni software è basato su un'architettura proprietaria, pilastro della trasformazione digitale della banca.

Questa architettura si distingue per la sua modularità e capacità di adattamento, consentendo la realizzazione di applicazioni bancarie che garantiscono un'esperienza utente omogenea e un'introduzione agile di nuovi prodotti e servizi digitali, in particolare fornisce molteplici framework che si focalizzano sugli elementi chiave di sviluppo di applicazioni modulari come: Frontend e interazione con gli utenti, Backend e logica applicativa, applicazioni batch e comunicazione asincrona ad eventi. Tale architettura è in continua innovazione, assicurando elevati standard di affidabilità e scalabilità, in questo modo il cliente è in grado di rispondere con efficacia alle sfide e di dimostrarsi in costante evoluzione, mantenendo un equilibrio ottimale tra sicurezza, efficienza operativa e capacità di innovazione tecnologica.

2.2.1 Framework utilizzati per il Backend

Nel contesto aziendale del cliente si richiede l'adozione di soluzioni tecnologiche altamente scalabili, sicure e manutenibili per supportare la vasta gamma di servizi bancari offerti. Lo sviluppo di applicazioni web robuste è cruciale per garantire un servizio efficiente agli utenti e la gestione ottimale delle operazioni interne. Il cliente

propone l'utilizzo di un framework proprietario per la realizzazione del backend di tali applicazioni.

Tale framework ha come base Spring Boot, consolidato per lo sviluppo di applicazioni Java, estendendolo e ottimizzandolo in modo tale che sia piú specifico per il contesto aziendale.

2.2.1.1 Spring boot

Spring Boot [5] è un framework che si distingue per la sua capacità di semplificare e accelerare lo sviluppo di applicazioni Java, è caratterizzato per ridurre drasticamente la quantità di configurazione manuale necessaria per avviare e mantenere un'applicazione, fornendo allo stesso tempo una serie di strumenti potenti per lo sviluppo, la gestione e la distribuzione del software. Questo paragrafo esamina in dettaglio i vantaggi e gli svantaggi dell'adozione di Spring Boot, fornendo una panoramica delle sue caratteristiche.

I vantaggi dell'utilizzo di Spring-boot per una applicazione java si possono riassumere in: facilità nella configurazione, permettendo mediante dei file appositi di configurazione di automatizzare questo processo riducendo drasticamente i tempi di configurazione. Dipendenze già presenti in particolari set chiamati starter che racchiudono una serie di funzionalità in modo da non dover configurare manualmente ogni singola dipendenza e un alto supporto per i microservizi permettendo la comunicazione tra di essi mediante diversi metodi tra cui API e broker di messaggi.

I principali svantaggi possono essere invece: la scarsa "visibilità", le configurazioni automatiche nascondono la complessità interna dell'applicazione e può essere difficile comprendere il funzionamento sottostante, un eccessivo carico di dipendenze e difficoltà nel configurare in modo personalizzato l'applicazione.

In conclusione, Spring Boot è una scelta eccellente per chi cerca di accelerare lo sviluppo di applicazioni Java riducendo il tempo speso in configurazioni manuali e fornendo un set completo di strumenti per la produzione.

Figura 2.3: Spring Boot logo

2.2.1.2 Panoramica framework proprietario

Il framework utilizzato per lo sviluppo del Backend delle applicazioni del cliente si basa sulla creazione di due tipologie di microservizi:

- **Backend for Frontend (BE4FE)**: questi microservizi gestiscono la comunicazione diretta con i client, gli input di questi microservizi sono le informazioni ricevute dal frontend e dal contesto della sessione specifica del canale di comunicazione.
- **Core**: microservizi che gestiscono la logica di business centrale dell'applicazione, spesso invocati dai BE4FE o da altri Core.

Esiste poi il Session Manager, con cui comunicano i BE4FE, che facilita la gestione del contesto delle sessioni e assicura che le informazioni rilevanti degli utenti siano propagate correttamente.

Il ciclo di vita di una richiesta è orchestrato dal Controller, che gestisce il flusso dei dati e consente l'esecuzione di varie operazioni, tra cui:

- Utilizzo di annotazioni per assicurare che i dati in ingresso rispettino i criteri di validità predefiniti.
- Invocazione del metodo execute di un oggetto Command, che decapsula la logica di business e gestisce la trasformazione dei dati.
- Generazione delle risorse attraverso il pattern assembler.
- Qualsiasi altra operazione custom definita per soddisfare i requisiti specifici dell'applicazione.

Il pattern Command è particolarmente importante poichè separa chiaramente la gestione dei dati dalla logica di business, migliorando la leggibilità e la manutenibilità del codice.

L'adozione di questo framework offre una soluzione flessibile e scalabile, capace di soddisfare le esigenze di complessi sistemi nel contesto aziendale.

La chiara separazione delle responsabilità, combinata con una robusta gestione del ciclo di vita delle richieste, consente di creare applicazioni affidabili e mantenibili. Si utilizzano pattern consolidati e best practices che possono essere facilmente integrate in qualsiasi progetto in modo da standardizzare e rendere omogeneo il ciclo di sviluppo di una web application nel contesto aziendale bancario.

2.2.2 framework utilizzati per il Frontend

In questa sezione vengono analizzati gli strumenti di sviluppo frontend che assumono un ruolo cruciale, per ottenere interfacce utente efficienti, modulari e facilmente manutenibili, capaci di operare su diversi canali.

In questo ambito, l'architettura del cliente propone un framework di Frontend che si basa su Angular.

2.2.2.1 Angular

Angular [6] è un framework frontend open-source sviluppato da Google, progettato per facilitare la creazione di applicazioni web single-page (SPA) attraverso un approccio modulare e component-based, esso offre una struttura ben definita che permette l'organizzazione del codice in moduli, componenti, servizi e direttive.

Angular funziona attraverso un sistema di data binding bidirezionale, che mantiene il modello dati e la vista sincronizzati automaticamente, riducendo la necessità di manipolare manualmente il DOM.

Utilizza un sistema di componenti altamente riutilizzabili, che rappresentano singole unità dell'interfaccia utente, ciascuna con la propria logica e struttura inoltre, offre servizi per gestire la logica di business e l'interazione con le API, e router per la gestione della navigazione tra le diverse viste.

I principali vantaggi nell'adozione di tale framework sono: la modularità e riutilizzabilità, in quanto il codice si suddivide in componenti e moduli riutilizzabili migliorando così la manutenibilità e riducendo blocchi di codice duplicati. Data binding bidirezionale che permette l'automatizzazione dell'aggiornamento della vista in risposta ai cambiamenti del modello, in modo da semplificare la gestione delle interfacce utente. Angular poi dispone di una serie di strumenti integrati che riducono la necessità di dipendenze esterne e semplificano lo sviluppo e gode inoltre di un ampio ecosistema di librerie e di una comunità attiva. Gli svantaggi nell'utilizzo di tale framework sono: l'alta complessità con una curva di apprendimento ripida e una relativa pesantezza rispetto ad altre soluzioni di Frontend.

In conclusione, Angular rappresenta una scelta robusta per lo sviluppo di applicazioni web complesse e scalabili, offrendo una vasta gamma di funzionalità integrate e un'elevata manutenibilità del codice.

Figura 2.4: Angular logo

2.2.2.2 Panoramica framework proprietario

Il framework realizzato dal cliente è stato progettato per supportare la creazione di applicazioni web che possano essere facilmente portabili su diverse piattaforme e canali, garantendo contemporaneamente una User Experience uniforme e coerente.

Una delle principali caratteristiche di questo framework è la modularità, che consente agli sviluppatori di suddividere le applicazioni in moduli indipendenti e riutilizzabili, questi sono poi implementati seguendo una architettura plug and play, permettendo una facile integrazione e personalizzazione delle applicazioni.

Inoltre, il framework supporta la creazione automatica dello scheletro dell'applicazione, semplificando il processo di sviluppo e riducendo il tempo necessario per avviare nuovi progetti.

Si compone di elementi architetturali di base, noti come Architectural Core Components, ai quali si affiancano componenti pluggable e moduli funzionali

Questi elementi, distribuiti in diverse categorie, collaborano per offrire un'architettura solida e flessibile che si adatta facilmente alle esigenze specifiche delle varie applicazioni.

La libreria di widget fornita dal framework permette di mantenere un'esperienza utente omogenea, indipendentemente dal dispositivo o dal canale utilizzato.

Inoltre risulta importante l'integrazione di questo framework con le pratiche DevOps, consentendo di implementare e gestire i propri moduli in modo efficiente attraverso l'intero ciclo di vita dello sviluppo software.

In sintesi, questa tecnologia rappresenta una soluzione avanzata e versatile per lo sviluppo frontend per le applicazioni dell'organizzazione, fornendo strumenti e metodologie che favoriscono la scalabilità, la riusabilità e la coerenza delle applicazioni su più canali e dispositivi.

2.2.3 framework per lo sviluppo di Batch

La gestione efficiente dei processi batch è cruciale per il trattamento di grandi volumi di dati e per l'automazione di task ripetitivi e complessi. Anche in questo caso L'architettura del cliente propone l'uso di un framework, progettato per semplificare ulteriormente lo sviluppo di processi batch all'interno dell'ambiente aziendale, basato sull'adozione di Spring Batch.

2.2.3.1 Spring Batch

Spring Batch [7] è un framework open-source parte dell'ecosistema Spring, progettato per supportare la gestione e l'esecuzione di processi batch in modo robusto e scalabile. È particolarmente adatto per applicazioni enterprise che devono elaborare grandi volumi di dati in modo efficiente e affidabile.

I principali componenti del framework sono i seguenti:

- Job e Step: Un Job in Spring Batch è l'entità principale che definisce un processo batch. È composto da una sequenza di Step, che rappresentano singole unità di lavoro all'interno del job. Ogni step esegue una parte specifica della logica di business e può essere configurato per elaborare dati utilizzando due modelli principali: Chunk-oriented processing e Tasklet.

- JobRepository, componente centrale di Spring Batch responsabile della persistenza dello stato del job. Esso conserva i metadati relativi all'esecuzione dei job e degli step, permettendo di monitorare il progresso, di gestire ripristini e riavvii e di eseguire analisi post-esecuzione.

- JobLauncher, il componente che avvia i job batch. È responsabile di passare i parametri al job, avviare il job e restituire lo stato dell'esecuzione. Il JobLauncher è tipicamente configurato in modo che possa essere chiamato da vari contesti, come un'applicazione web, un processo pianificato o un'applicazione standalone.

- ItemReader, ItemProcessor, e ItemWriter: questi componenti sono responsabili della lettura dei dati, trasformazione dei dati e scrittura dei dati in una destinazione

L'utilizzo di Spring Batch è vantaggioso in quanto gestisce in modo efficente i dati mediante il chunk processing, processando cioè i dati in blocchi (chunk) migliorando le performance delle transazioni. La divisione in moduli ne consente il riutilizzo e la combinazione per processi batch complessi, facilitando la manutenzione e l'evoluzione del codice aumentando la scalabilità del sistema. Offre inoltre un supporto nativo a molteplici tipi di sorgenti e destinazione dei dati.

Figura 2.5: Spring batch logo

2.2.3.2 Panoramica framework proprietario

Il framework del cliente per lo sviluppo Batch, introduce una serie di estensioni e astrazioni specifiche per le esigenze aziendali, con l'obiettivo di migliorare l'efficienza nello sviluppo, nella gestione e nel rilascio di job batch complessi.

Le caratteristiche principali di questo framework sono le seguenti:

1. **Struttura semplificata del Codice**: é progettato per ridurre la complessità tipica dello sviluppo di processi batch in Java. Esso fornisce un insieme di astrazioni e componenti predefiniti che semplificano la configurazione e l'implementazione dei job. Questo approccio permette di concentrarsi sulla logica di business piuttosto che sui dettagli tecnici della configurazione del framework, inoltre offre template predefiniti per la creazione di job e step, che riducono la necessità di scrivere codice boilerplate e assicurano che tutti i job siano conformi agli standard aziendali.

2. Una delle innovazioni chiave è la **gestione strutturata delle release**. Questo processo prevede il versionamento dei job batch su repository come BitBucket e il deployment su piattaforme come Layer Batch.

3. **Integrazione con Spring Batch e altri strumenti aziendali**: pur essendo basato su Spring Batch, aggiunge ulteriori funzionalità per una migliore integrazione con l'infrastruttura IT esistente, tra cui il supporto per strumenti di

monitoraggio, pianificazione e gestione delle operazioni batch, si integra nativamente con Tivoli Workload Scheduler (TWS), orchestratore dei job batch, in modo da garantire che questi vengano eseguiti secondo le finestre temporali prestabilite, migliorando la gestione delle operazioni. Sono inclusi anche strumenti per il monitoraggio dettagliato dei job, permettendo di tracciarne anche lo stato, raccoglierne le metriche di performance e gestirne i log di esecuzione.

4. **Supporto per operazioni complesse**: è progettato per supportare configurazioni complesse, si possono facilmente configurare i job con requisiti specifici, come la gestione delle transazioni, l'implementazione di logiche di fallback, o l'esecuzione di task condizionali.

Nell'utilizzo di questo framework si hanno i seguenti vantaggi:

- **Riduzione tempi di sviluppo**: si riduce significativamente la complessità dello sviluppo dei job batch grazie ai template forniti, permettendo agli sviluppatori di concentrarsi sulla logica di business, accelerando il ciclo di sviluppo e migliorando la qualità del codice sorgente.

- **Scalabilità e adattabilità** : il framework è stato progettato per essere scalabile, con la possibilità di configurare job batch complessi e di gestire ambienti multitenant consentendo di rispondere rapidamente a nuove esigenze di business senza compromettere le prestazioni o la manutenibilità

- **Integrazione e supporto aziendale**: La stretta integrazione con strumenti aziendali come TWS e le funzionalità di monitoraggio avanzate forniscono un ambiente di esecuzione altamente controllato e monitorato. Questo non solo migliora l'affidabilità dei job, ma offre anche un supporto per l'identificazione e la risoluzione dei problemi.

Capitolo 3

Estrazione, analisi e caricamento dei dati

In questo capitolo si analizzano i processi di acquisizione, estrazione, trasformazione e caricamento (ETL) dei dati provenienti da diverse fonti verso un sistema centralizzato, dove i dati possono essere ulteriormente elaborati e analizzati.

La complessità e l'importanza di queste attività derivano dal volume e dalla varietà dei dati da gestire, nonchè dalla necessità di garantire l'integrità, la precisione e la tempestività delle informazioni contabili.

Per orchestrare questo processo, viene utilizzato Tivoli Workload Scheduler (TWS), cui compito è quello di eseguire delle applicazioni schedulate oppure unattended composte da una serie di job, i quali eseguono shell script, che a loro volta eseguono Batch, DataStage o query SQL per gestire le operazioni di estrazione, trasformazione e caricamento dei dati. Si esplora in dettaglio il processo approfondendo gli strumenti utilizzati e il contributo personale all'interno di alcune particolari scelte implementative.

3.1 Analisi degli strumenti utilizzati

Per la corretta realizzazione di questo processo, verrano utilizzati una serie di strumenti, descritti di seguito, e implementazioni SW che mirano a rendere il processo robusto, flessibile, scalabile e automatico.

3.1.1 IBM Tivoli Workload Scheduler

Il Tivoli Workload Scheduler (TWS) [8] è uno strumento fondamentale per l'automazione e la gestione dei job che compongono il flusso di data ingestion.

TWS consente di orchestrare e pianificare l'esecuzione di diversi script e processi batch, garantendo che tutte le operazioni di estrazione e trasformazione dei dati vengano eseguite in modo coordinato e in tempi specifici.

I processi schedulati rappresentano l'utilizzo tipico di questo strumento, ma è possibile anche eseguire un'applicazione TWS in modo unattended, eseguendola cioè in un determinato momento on-demand.

3.1.1.1 Benefici dell'utilizzo di TWS

I benefici derivanti dall'utilizzo di TWS sono:

- **Automazione e Affidabilità**: TWS permette di automatizzare l'intero processo di data ingestion, riducendo al minimo l'intervento manuale e garantendo che i job vengano eseguiti in modo affidabile e secondo la pianificazione prestabilita.
- **Gestione degli errori**: In caso di fallimento di un job, TWS può essere configurato per inviare notifiche, riavviare il job o eseguire azioni di fallback, migliorando la resilienza dell'intero sistema.
- **Scalabilità**: TWS supporta la gestione di flussi complessi e su larga scala, permettendo di aggiungere facilmente nuovi job o modificare le pianificazioni esistenti senza compromettere l'efficienza del sistema.

3.1.2 Utilizzo di shell script

Una volta avviati da TWS, i job eseguono delle shell script, situate nel batch-layer, che rappresentano il cuore del processo di data ingestion. Questi script sono responsabili di orchestrare l'esecuzione dei componenti Batch, DataStage o delle query SQL necessarie per l'estrazione e la trasformazione dei dati.

3.1.2.1 Benefici dell'utilizzo di shell script

L'utilizzo di shell script comporta i seguenti benefici:

- **Flessibilità**: Le shell script offrono un alto grado di flessibilità, permettendo di integrare facilmente diversi strumenti e tecnologie nel processo di data ingestion. Possono essere utilizzate per eseguire comandi di sistema, lanciare applicazioni batch o richiamare API esterne.
- **Semplicità di gestione**: Le shell script sono facili da scrivere e modificare, permettendo di adattare rapidamente il processo di data ingestion alle nuove esigenze o modifiche nel flusso dei dati.
- **Integrazione con altri sistemi**: Le shell script possono interagire con vari sistemi operativi e piattaforme, rendendole uno strumento versatile per coordinare le diverse fasi del processo di ETL.

3.1.3 IBM DataStage

IBM DataStage [9] è uno strumento ETL potente e versatile, utilizzato per eseguire trasformazioni complesse sui dati prima che vengano caricati nel sistema di destinazione.

DataStage viene utilizzato per processare grandi volumi di dati e applicare trasformazioni che richiedono elaborazioni sofisticate, come aggregazioni, join, o calcoli avanzati.

3.1.3.1 Benefici utilizzo IBM DataStage

IBM DataStage fornisce i seguenti benefici:

- DataStage è progettato per gestire grandi volumi di dati e supportare trasformazioni complesse, rendendolo ideale per operazioni ETL su larga scala.
- Grazie alla sua architettura parallela, DataStage può essere scalato facilmente per gestire un numero crescente di dati, mantenendo elevate prestazioni di elaborazione.
- DataStage offre un'interfaccia grafica intuitiva per la progettazione dei flussi ETL, permettendo agli sviluppatori di costruire e gestire pipeline di dati complesse con meno codice e maggiore visibilità sui processi.

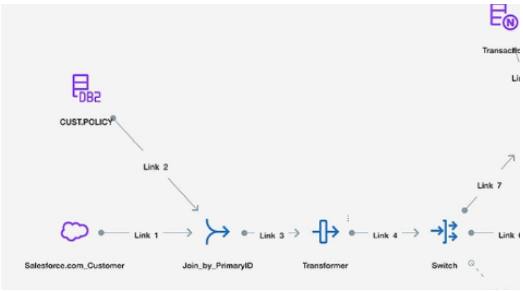

Figura 3.1: Pipeline di IBM DataStage di esempio

3.1.4 Esecuzione query SQL mediante SQL*Plus

Una componente fondamentale del processo è l'esecuzione di query SQL per estrarre, trasformare e caricare i dati nei database esistenti.

In molti casi, questo viene realizzato utilizzando SQL*Plus, uno strumento di riga di comando per l'interazione con i database Oracle.

SQLPlus è un'interfaccia potente per eseguire comandi SQL, PL/SQL e blocchi anonimi direttamente su un database Oracle. Quando viene eseguito all'interno di shell script, SQLPlus consente di automatizzare operazioni complesse, come l'estrazione dei dati, la trasformazione e il caricamento nei sistemi di destinazione.

3.1.4.1 Benefici utilizzo SQLPlus

I benefici nell'utilizzo di SQLPlus sono:

- L'integrazione di SQL*Plus negli shell script permette di automatizzare completamente il processo di estrazione e trasformazione dei dati. Gli script possono essere pianificati tramite TWS, garantendo che le operazioni vengano eseguite in modo affidabile e secondo una sequenza predefinita.

- SQL*Plus consente di eseguire qualsiasi comando SQL o PL/SQL, offrendo la flessibilità necessaria per implementare trasformazioni complesse o operazioni di manutenzione sui dati direttamente a livello di database.

- **Interazione diretta con il database**: SQL*Plus permette un'interazione diretta con il database Oracle, sfruttando appieno le capacità del motore SQL per l'ottimizzazione delle query e l'efficienza delle operazioni di manipolazione dei dati.

- Utilizzando shell script per orchestrare l'esecuzione di SQL*Plus, è possibile centralizzare e standardizzare la gestione delle operazioni di ETL, facilitando la manutenzione e l'aggiornamento dei processi di data ingestion.

3.1.5 Utilizzo di Batch

Le shell script durante la loro esecuzione possono richiamare l'esecuzione di un componente Batch sviluppato con il framework del cliente, decidendo quali job eseguire e con quali parametri. L'adozione di questa tecnologia per la gestione dei processi batch offre un set completo di strumenti e funzionalità che superano le capacità dell'esecuzione diretta di query tramite SQLPlus, soprattutto quando si tratta di gestire processi complessi e scalabili.

3.1.5.1 Vantaggi nell'utilizzo di Batch

Come già descritto, l'adozione del framework del cliente per la gestione dei processi batch rappresenta un notevole vantaggio per soluzioni robuste, scalabili e facili da gestire. La semplificazione del processo di sviluppo, unita alla capacità di gestire complessi flussi decisionali e alla stretta integrazione con l'infrastruttura aziendale, rende questo strumento ideale per l'elaborazione di grandi volumi di dati.

I vantaggi derivanti dall'uso di questa tecnologia, come l'efficienza operativa, la gestione strutturata delle release e la scalabilità, assicurano che i job batch possano essere eseguiti in modo affidabile e in linea con le esigenze aziendali in continua evoluzione.

3.2 Modello logico dei dati

Il modello logico dei dati adottato nel sistema si basa su un'infrastruttura Oracle e riflette una suddivisione funzionale volta a ottimizzare la gestione e l'elaborazione delle movimentazioni contabili.

Questo modello è strutturato su due grandi database, ciascuno con un compito specifico, al fine di garantire efficienza operativa, scalabilità e una gestione accurata dei dati.

La scelta di Oracle come piattaforma garantisce affidabilità e prestazioni elevate, supportando le complesse esigenze di gestione dei dati dell'intero sistema contabile.

Il primo database agisce come archivio principale e contiene l'insieme completo delle movimentazioni contabili, provenienti dai sistemi legacy aziendali. Questo database è il punto centrale in cui confluiscono tutte le transazioni contabili, consentendo di mantenere uno storico dettagliato e completo delle operazioni. Le principali tabelle seguono uno schema ben definito e consolidato, adottato all'interno del contesto aziendale. L'architettura dati prevede l'utilizzo di due tipi di tabelle principali, ognuna con una funzione specifica.

- La prima tabella, denominata "work", funge da area temporanea per i dati grezzi che vengono acquisiti durante la fase iniziale di data ingestion. Qui i dati vengono inseriti senza particolari elaborazioni, rappresentando un'istantanea delle informazioni appena ricevute. Successivamente, in base a regole di business precise e validazioni specifiche, i dati presenti nella tabella "work" vengono elaborati e trasformati.

- A partire dai dati validati e processati nella tabella "work", viene popolata la tabella "on-line", che rappresenta il dataset attivo. Questa è la tabella effettivamente interrogabile dalle applicazioni e dai sistemi per le operazioni quotidiane. Essendo costantemente aggiornata, la tabella "on-line" contiene i dati più recenti e rilevanti, pronti per essere utilizzati nei processi decisionali e di elaborazione.

Per particolari tabelle che ricevono un'elevata mole di dati, viene definita una tabella dedicata allo storico, per garantire la conservazione di informazioni storiche e supportare esigenze di audit e tracciamento. Questa contiene i dati che non sono più utilizzati nelle operazioni quotidiane ma devono essere mantenuti per conformità e analisi retrospettive. La tabella di storico è partizionata seguendo criteri specifici, come la suddivisione per periodi temporali o tipologie di dati, per garantire un accesso rapido e ottimizzato anche quando il numero di record è molto elevato. Questo approccio consente di bilanciare la necessità di conservare grandi quantità di dati storici con le prestazioni richieste per eventuali interrogazioni e analisi future.

Per la realizzazione di questa particolare struttura si utilizzano pipeline di IBM DataStage che consentono le corrette fasi di data ingestion delle suddette tabelle e le corrette operazioni di trasformazioni del dato.

Il secondo database è stato concepito per soddisfare le esigenze specifiche degli ambiti di sviluppo richiesti da questo progetto. A differenza del primo, la sua struttura

è più snella e semplificata, non seguendo lo schema complesso di tabelle "work", "on-line", e di storico utilizzato per la gestione delle movimentazioni contabili.

In questo caso, ogni tabella gestisce direttamente i dati di interesse senza distinzioni tra fasi di ingestione o attivazione. La scelta di non includere tabelle di storico riflette il fatto che la quantità di dati trattata in questo contesto è significativamente inferiore e non richiede una gestione separata dei dati storici. Tuttavia, pur essendo più semplice, la struttura di questo database è stata attentamente ottimizzata per garantire performance elevate.

Le tabelle sono state migliorate attraverso l'uso di indici e partizionamenti basati su criteri specifici, come chiavi univoche o parametri temporali, in modo da garantire tempi di risposta rapidi durante le fasi di interrogazione. Queste ottimizzazioni sono state implementate per massimizzare l'efficienza delle query, riducendo al minimo i tempi di accesso ai dati e migliorando le prestazioni complessive del sistema, anche in presenza di un numero crescente di richieste simultanee.

In sintesi, questo database è progettato per garantire una gestione dei dati più agile, senza sacrificare la velocità e l'efficienza, rispondendo in maniera efficace alle specifiche esigenze del progetto.

3.3 Realizzazione pratica

In questa sezione, dopo una serie di descrizioni sugli strumenti utilizzati e la gestione delle tabelle delle base di dati coinvolta, viene affrontata la realizzazione pratica di alcuni requisiti dei diversi ambiti di sviluppo di cui mi sono occupato personalmente, in cui si va ad analizzare ogni scelta intrapresa.

Il fulcro operativo alla base di tutti gli ambiti del progetto, risiede nei cosiddetti "giri contabili", momenti specifici della giornata in cui avviene l'estrazione e l'elaborazione dei dati rilevanti. Poichè queste operazioni devono essere eseguite in orari prestabiliti su base giornaliera, la fase di data ingestion è gestita in maniera completamente automatizzata attraverso l'orchestrazione di processi pianificati utilizzando Tivoli Workload Scheduler (TWS).

Questo approccio assicura un flusso continuo e regolare di acquisizione dei dati, garantendo che le informazioni necessarie siano sempre disponibili nel sistema in modo accurato.

Ciascuna di queste applicazioni esegue in un determinato ordine una serie di job responsabili dell'esecuzione delle rispettive shell script, questo rappresenta il cuore del processo, in cui i dati vengono estratti ed elaborati.

Per l'alimentazione del database principale, che raccoglie tutte le movimentazioni contabili derivate dai sistemi legacy, vengono implementate una serie di pipeline avanzate basate su IBM DataStage. Questa soluzione consente di eseguire operazioni complesse su un ampio volume di dati, garantendo efficienza e scalabilità.

Il processo di data ingestion è articolato in diverse sotto-fasi, ognuna delle quali svolge un ruolo cruciale nell'integrità e nell'ottimizzazione dei dati. In particolare, si prevede il popolamento delle tabelle di lavoro ("work"), che fungono da area di staging per la manipolazione preliminare dei dati.

Inoltre, in scenari in cui è necessario trasferire informazioni nella tabella storica, viene eseguito il troncamento delle tabelle online, assicurando che i dati obsoleti vengano rimossi in modo sicuro e che il sistema rimanga performante.

Infine, la pipeline culmina nel popolamento della tabella online, garantendo che i dati aggiornati siano sempre disponibili per le operazioni quotidiane e per le analisi future.

3.3.1 Sorgenti del dato

La base nella realizzazione di nuovi processi ETL parte sempre dalla comprensione dell'origine dei dati e di come ottenere gli stessi. Di seguito si descrivono alcune scelte, relative gli ambiti di sviluppo, in cui sono stato coinvolto personalmente per il corretto ottenimento dei dati di interesse.

3.3.1.1 Ricezione nuovi crediti

L'applicativo, per quanto concerne l'ambito di sviluppo della gestione dei crediti, deve essere predisposto, quotidianamente, alla ricezione di crediti provenienti da un partitario e dovrà gestirne l'elaborazione in base a specifiche regole di gestione dei crediti come la suddivisione in annualità e l'aggiornamento degli importi.

Mediante la collaborazione con il partitario responsabile della trasmissione del file contenente i crediti, si è scelto di implementare un'applicazione TWS che viene scatenata dalla ricezione del suddetto file.

In particolare, il file transita sul batch layer, nella cartella di input, in cui punta la shell invocata dal job dell'applicazione.

Il focus di questa operazione è la resilienza, in quanto il processo deve minimizzare gli errori che possono derivare da una lettura sbagliata di una particolare riga del file o simili. In questo caso le difficoltà maggiori si sono riscontrate proprio nella scrittura dello script responsabile di: leggere il file in input e inserire i dati nelle tabelle di anagrafica dei crediti, aggiornando altre tabelle collegate nel caso ce ne sia bisogno mediante opportune regole.

Il mio contributo in questo processo è stato proprio sulla scrittura di questi script, assicurandomi che tutti i requisiti del cliente fossero rispettati.

Per la lettura del file si usano i comandi integrati in bash e la struttura del file è stata scelta in collaborazione con i responsabili dell'invio.

Una volta ottenuti i dati dal file, si procede mediante query SQL, utilizzando SQL*Plus, ad aggiornare massivamente le tabelle del database di interesse. In particolare sono coinvolte le tabelle di anagrafica del credito che, mediante il suo id univoco, è collegato ad altre tabelle che gestiscono altre informazioni come le suddivisioni annuali dello stesso. Per questo motivo se ad esempio si riceve un importo diverso per un credito già esistente, occorre aggiornare anche tutte le tabelle di interesse.

Dopo questa prima fase di acquisizione, l'applicazione TWS esegue altri job responsabili del controllo e validazione dei dati inseriti nelle apposite tabelle, questi sono fondamentali per garantire che il processo di ricezione sia avvenuto correttamente e nel caso segnalare le squadrature rilevate su particolari id dei crediti in modo tale da migliorare l'analisi dell'anomalia.

Infine l'ultimo job è responsabile di richiamare una shell che ha il compito di inviare una comunicazione mediante mail che il processo è ultimato, segnalando nel caso le squadrature incontrate. Questo è possibile sempre mediane strumenti disponibili sul Sistema operativo e mediante comandi bash.

Questa scelta implementativa rappresenta un modus operandi consolidato all'interno dell'organizzazione, utilizzato in svariati altri contesti, di conseguenza non è stato chiesto un particolare sforzo sull'utilizzo di una tecnologia in particolare ma, sulla corretta realizzazione del processo in modo tale che non vengano scatenati errori e che nel caso siano parlanti e presenti nelle comunicazioni inviate, in modo da intervenire tempestivamente su appositi crediti.

Il mio contributo, quindi è stato quello di realizzare gli script, in modo da prevenire gli errori con una scrittura semplice ma efficace, seguendo le linee guida dell'organizzazione.

3.3.1.2 Estrazione saldi e movimenti

L'ambito di sviluppo dei fondi utilizzabili in caso di alto rischio, si lega alle attività degli utenti su un altro strumento su cui vengono caricate le informazioni contabili dei fondi, in particolare saldi e movimenti, alimentando il database principale sopra elencato.

Questa attività provoca una sorta di "dualismo di responsabilità" del dato che viene gestito da diverse fonti e deve rimanere accurato e preciso durante tutte queste attività.

Durante la visualizzazione dei fondi nella piattaforma web realizzata, gli utenti necessitano di leggere le informazioni dei saldi e dei movimenti dei fondi, presenti nelle tabelle del database principale.

Nonostante l'ottimizzazione delle tabelle di interesse, indicizzate su particolari colonne, lo scan di queste risulta particolarmente oneroso e le interrogazioni su questo database rappresentano dei colli di bottiglia per le operazioni "on'line".

Il mio compito in questo caso è stato quello di ottimizzare il processo, in modo da rendere la piattaforma più responsiva e migliorare l'esperienza utente.

Per riuscire in questi intenti, le tabelle da cui leggere le informazioni dei saldi e dei movimenti dovevano essere ovviamente più piccole e snelle delle principali, contenendo solo le informazioni di interesse dei fondi mostrati dalla piattaforma.

L'idea realizzativa si basa sull'estrazione dei saldi e movimenti di interesse dal database principale e il successivo inserimento di questi dati sul database secondario aggiornando le nuove tabelle che risultano più snelle e leggere. In questo modo le interrogazioni su queste tabelle sono molto più veloci e consentono un tempo di risposta quasi istantaneo.

I dati però, devono essere costantemente aggiornati in quanto in ogni momento può essere eseguita una transazione che rappresenta una modifica sui saldi e movimenti contabili del fondo.

La scelta ideale, sarebbe stata la realizzazione di una architettura event-driven che alla contabilizzazione di nuovi saldi e movimenti aggiornasse in tempo reale le nuove tabelle di interesse. Questa soluzione non è stata adottata in quanto necessitava di modifiche software ad applicativi non di competenza del team.

La scelta implementativa è ricaduta su un processo schedulato che ogni trenta minuti aggiorna le nuove tabelle di interesse dopo aver estratto i dati dalle tabelle

del database principale mediante l'utilizzo di chiavi di raccordo che caratterizzano i fondi stessi.

Poichè non interessa lo storico del saldo e dei movimenti, i nuovi dati aggiornano i precedenti sostituendoli senza generare nuovi record nelle tabelle, in questo modo rimangono snelle e leggere costituendo una sorta di cache in cui la piattaforma web legge i dati.

Per la realizzazione di questo processo, si è adottata una applicazione TWS schedulata, come detto, ogni trenta minuti che esegue il job responsabile di eseguire la shell che estrae i dati ed aggiorna le nostre tabelle di interesse.

Nella realizzazione della shell, si potevano utilizzare svariate tecnologie per la realizzazione del flusso dei dati. In particolare si poteva richiamare un job di un componente Batch, oppure eseguire direttamente le query mediante SQL*Plus sia per l'estrazione che per l'aggiornamento dei dati.

Inoltre si potevano conservare le estrazioni eseguite in un file, processato succesivamente da un altro job e di conseguenza da un'altra shell cui compito, sarebbe stato l'aggiornamento delle tabelle rispetto i dati conservati nel file. Con questo approccio si sarebbe garantito il principio di single responsability di ogni shell.

Poichè le regole di estrazione dei dati dal database principali erano ben delineate e semplici, visto il perimetro ristretto dei dati dell'ambito di sviluppo, la scelta è ricaduta sull'implementazione di una singola shell che mediante SQL*Plus effettui l'estrazione dei dati e l'aggiornamento successivo delle nuove tabelle. Infatti si preferisce un componente Batch, sviluppato con il framework proprietario del cliente, quando le regole di estrazione e processamento dei dati sono molto complesse tanto da voler utilizzare le strutture dati e l'orientamento agli oggetti di Java. Inoltre il processo deve essere veloce e anche in questo caso altamente resiliente, quindi la suddivisione in più elaborazioni con l'utilizzo di molteplici job poteva aumentare errori e rallentare l'esecuzione generale.

Infine alla conclusione del processo, viene alimentata anche una tabella che conserva tutti i processi effetuati, in modo da esporre agli utenti la data e ora dell'ultimo aggiornamento dei saldi.

Dopo la realizzazione di questo meccanismo, le prestazioni dell'applicativo, in particolari sezioni che esponevano i dati prima contenuti solo nel database "legacy", sono migliorate permettendo una migliore esperienza utente.

La figura 3.2 mostra lo schema delle soluzioni implementate

3.3.2 Alimentazione database di interesse

Come descritto nelle sezioni precedenti, il database principale, contenente tutti i dati delle movimentazioni contabili provenienti dai legacy, si dimostra essere una delle principali fonti del dato di interesse per l'alimentazione del database in utilizzo dagli ambiti di sviluppo personalmente trattati.

Oltre alle attività descritte precedentemente per quanto riguarda le estrazioni e ricezione di paricolari dati, si eseguono quotidianamente una serie di processi che aggiornano costantemente il database per garantire i requisiti funzionali e le operatività degli utenti. In particolare questo avviene mediante due modalità principali:

1. Query di estrazione diretta dal primo database: In scenari dove le regole di estrazione sono relativamente semplici e non richiedono complessi processi di trasformazione, vengono utilizzate query SQL per trasferire i dati dal primo database. Queste query permettono di scaricare rapidamente i dati, mantenendo un'alta efficienza operativa e riducendo al minimo il carico computazionale.

2. Componenti Batch per regole complesse: Quando sono necessarie regole di elaborazione più articolate, che includono condizioni complesse o trasformazioni multiple, entra in gioco un componente Batch scritto in Java mediante il framework proprietario del cliente. Questo componente è progettato per gestire in modo efficiente logiche condizionali, elaborate trasformazioni dei dati e la gestione di flussi complessi. Grazie ad esso, è possibile automatizzare processi di ingestion che richiedono un'elaborazione avanzata dei dati, garantendo al contempo la coerenza e l'affidabilità del flusso di dati.

Oltre ai processi automatici orchestrati da TWS, le tabelle vengono costantemente aggiornate durante l'interazione degli utenti con la web application. Ogni azione significativa dell'utente che impatta sui dati comporta l'inserimento o l'aggiornamento diretto dei record nelle tabelle del database. Questo garantisce che le informazioni nel sistema siano sempre aggiornate in tempo reale, riflettendo le attività degli utenti e consentendo un flusso dati costante, sia a livello operativo che decisionale.

3.3.2.1 Utilizzo TWS unattended

In base al requisito del cliente, una particolare azione seppur necessiti di un'attività batch, può essere eseguita on-demand dall'utente e non essere per forza schedulata.

Dopo aver ricevuto un particolare requisito e dopo aver analizzato sia lo scenario schedulato che quello unattended, si procede nel fornire la soluzione più congeniale all'esigenza.

Nell'ambito di sviluppo riguardante i crediti deteriorati, ho contribuito attivamente nel realizzare un processo batch non schedulato ma unattended, responsabile, in modo simile a quanto descritto prima, di estrarre informazioni riguardanti le movimentazioni contabili di particolari crediti dal database principale e di riportarle su delle tabelle del database secondario, anche in questo caso, più snelle e leggere.

Questo processo inizia dall'utente che richiede l'estrazione dei dati di interesse in un particolare momento, tramite apposita C.T.A presente sulla piattaforma web realizzata. Questo scatena l'esecuzione di una API che è responsabile di invocare l'applicazione TWS in modo unattended.

In particolare, il backend utilizza un metodo esposto proprio dall'applicazione TWS stessa che è responsabile di "mettere a piano" l'applicazione e di conseguenza scatenarne l'esecuzione.

La scelta di un processo batch scatenato dall'applicazione e non di un metodo asincrono gestito dal backend stesso si basa sull'elevata mole di dati che bisogna estrarre dal database principale. L'attività asincrona porta comunque a un utilizzo di risorse sul componente che può provocare un'esperienza utente meno piacevole. L'esecuzione della TWS invece, è totalmente trasparente all'applicativo che può essere utilizzato senza cali di performance dall'utente.

Nella realizzazione del processo si è fatta attenzione a possibili errori da parte dell'utente o a schedulazioni contemporanee di utenti diversi. Infatti è inutile ripetere l'operazione nell'arco di brevi periodi e se un'estrazione precedente non è ancora terminata.

Per far fronte a questo problema, è stata costruita una tabella di storico delle esecuzioni che sono rappresentate da un particolare ID e tipologia e caratterizzate da uno stato. Se l'applicativo nota operazioni nello stato: IN CORSO, non permette la richiesta di nuove estrazioni di quella tipologia. Una volta terminato il processo, si aggiorna la relativa operazione impostando lo stato in: ESEGUITO.

Anche in questo caso si sono fatte delle scelte sulla scrittura delle shell responsabili delle estrazioni dei dati. Si possono fare tutte le considerazioni precedenti e anche in questo caso la scelta è ricaduta su una shell responsabile di estrarre i dati dal database principale ed aggiornare il secondario mediante query SQL scritte con SQL*Plus.

3.3.2.2 Gestione e monitoraggio TWS mediante DWC

Il Dynamic Workload Console (DWC) rappresenta l'interfaccia web unificata e strategica per la gestione e il monitoraggio del Tivoli Workload Scheduler (TWS).

DWC funge da punto di controllo operativo centrale per l'intera rete di scheduling. Questa console offre funzionalità avanzate di monitoraggio in tempo reale e gestione dei workload, permettendo di accedere ai piani di scheduling correnti, di prova o storici, e di monitorare le interdipendenze tra i vari job e job stream.

La mia attività quotidiana, si basa sull'utilizzo di questo strumento che consente di analizzare la bontà delle implementazioni scelte e di controllare la salute delle applicazioni schedulate cpn viste grafiche dettagliate, come la job stream view e la plan view, in cui si può visualizzare e analizzare l'avanzamento dei job, identificare eventuali colli di bottiglia e intervenire direttamente sui job e sulle loro dipendenze. Inoltre, si possono applicare criteri di filtro per navigare e modificare dinamicamente i contenuti dei piani, facilitando la risoluzione dei problemi e la pianificazione futura.

In particolare ogni giorno si analizzano i log e gli andamenti generali delle TWS, intervenendo nel caso in cui ci siano abend (abnormal end) ripetuti su particolari applicazioni.

Per risalire alla causa del problema, si sfruttano i log generati dalle shell da cui è possibile individuare il punto critico che ha scatenato l'anomalia, ripetere l'operazione in ambiente di staging e dopo opportune fasi di debug si può intervire, proponendo una fix risolutiva o allarmando gli owner della causa del problema.

3.4 Conclusioni

Il processo di ETL dei dati è una parte cruciale dell'intero progetto, essenziale per garantire che i dati siano correttamente estratti, trasformati e caricati nel sistema di destinazione. L'adozione di strumenti e strategie come Tivoli Workload Scheduler, shell script, Batch, IBM DataStage e query SQL permette di costruire un flusso di data ingestion robusto, flessibile e scalabile. Ogni componente contribuisce a ottimizzare l'efficienza e migliorare la manutenibilità, elementi fondamentali per il successo del progetto e per il supporto alle decisioni aziendali basate sui dati contabili.

Il modello dei database utilizzati, garantisce integrità e consistenza dei dati e flessibilità nella gestione di quelli attuali e storici, cruciali per il settore bancario. Tuttavia, introduce complessità di manutenzione, elevato consumo di risorse e crescenti requisiti

di archiviazione nel tempo, richiedendo un'attenta gestione e ottimizzazione. Queste problematiche saranno costantemente monitorate e rappresenteranno un'opportunità per migliorare continuamente il sistema, garantendo al contempo un elevato livello di qualità nella soluzione tecnologica adottata.

In sintesi, tutte le fasi del flusso dei dati sono state progettate per essere altamente automatizzate e flessibili, con la capacità di adattarsi a diversi livelli di complessità nei processi di estrazione e trasformazione dei dati, assicurando al contempo un aggiornamento continuo e in tempo reale degli stessi grazie alle interazioni con la web application.

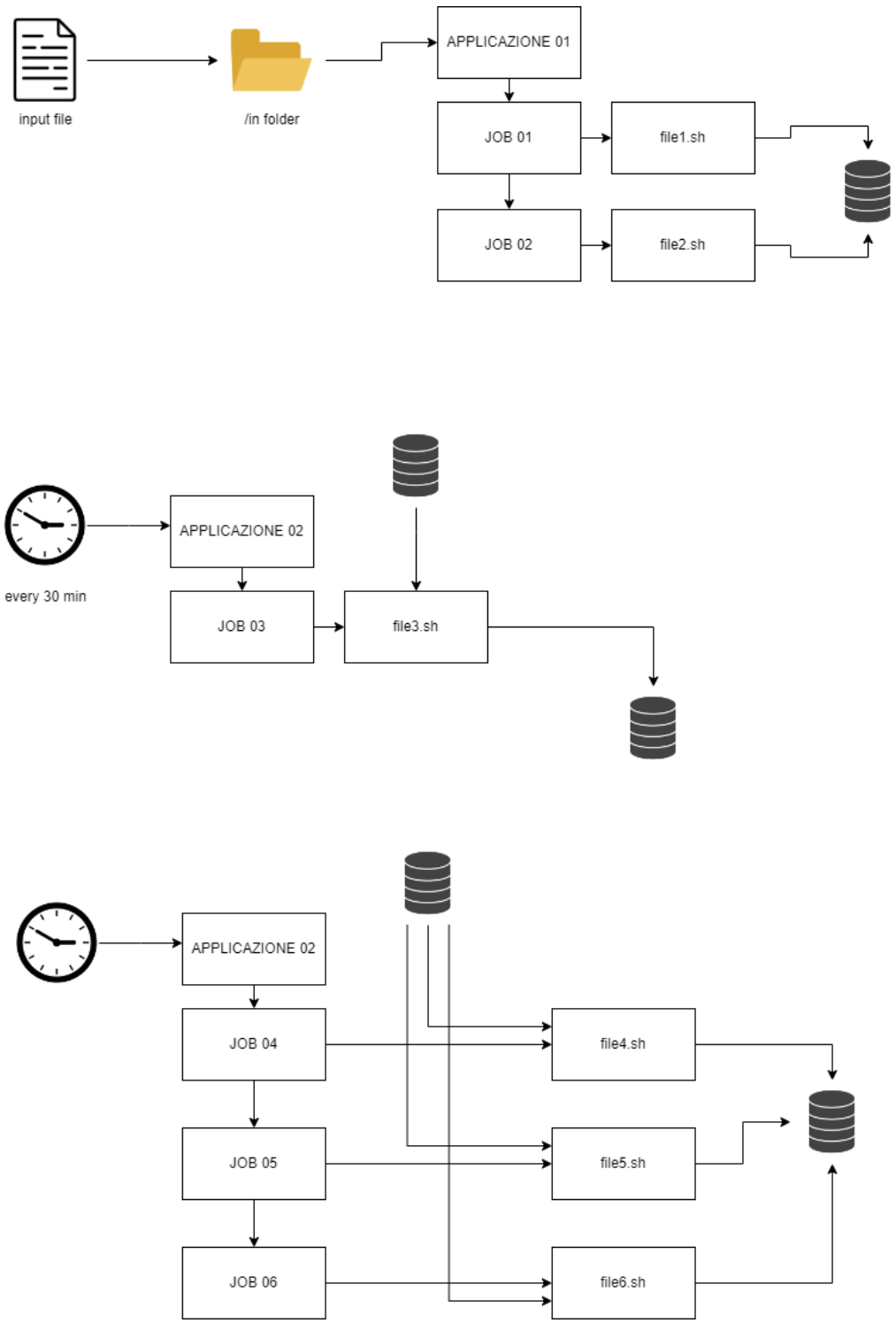

Figura 3.2: Implementazioni pipeline

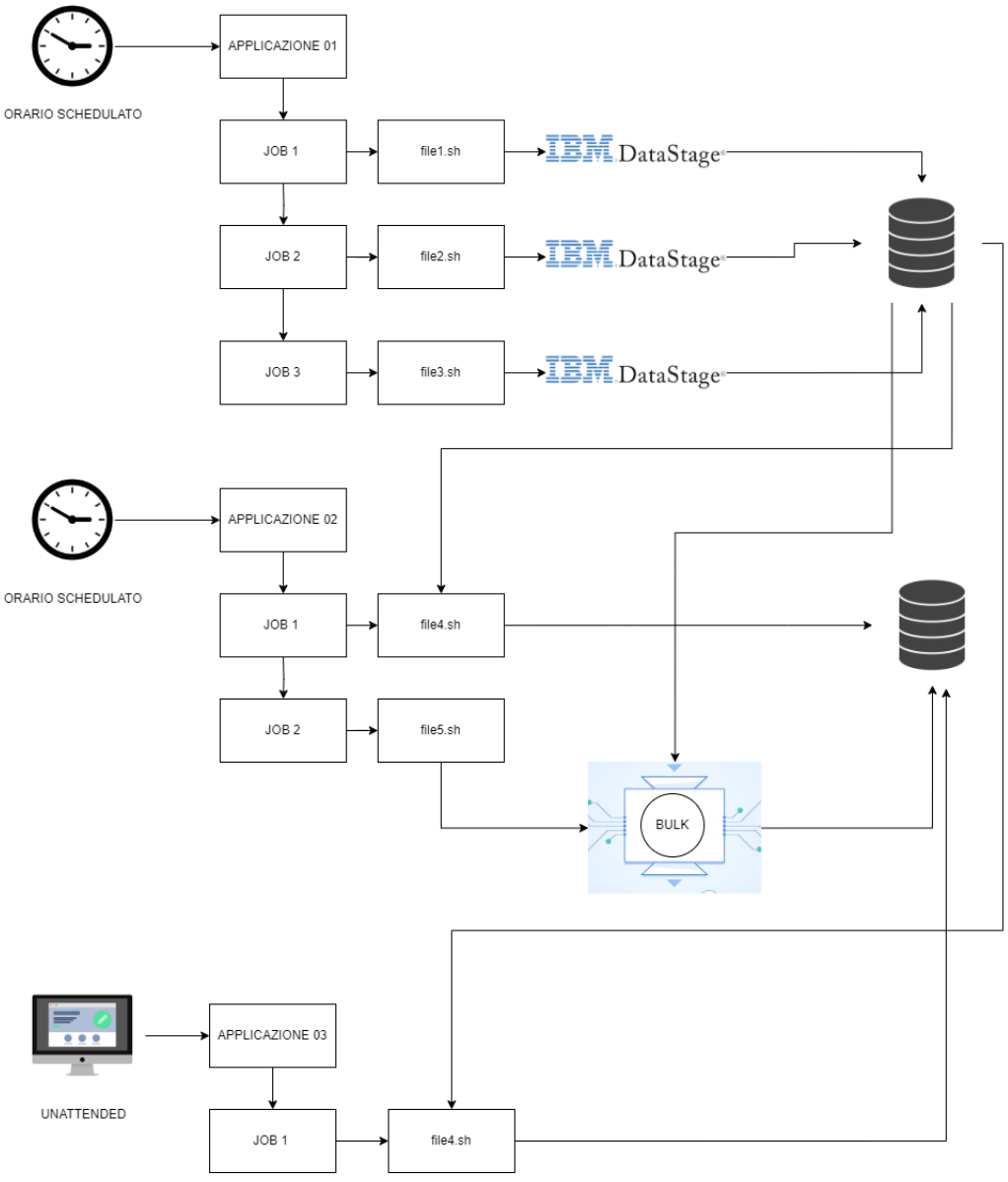

Figura 3.3: esempio di processo E.T.L realizzato

Capitolo 4

Sviluppo e gestione del ciclo di vita della web application

4.1 Introduzione

Il processo di sviluppo di un'applicazione web complessa, come quella descritta in questo progetto di tesi, richiede una pianificazione accurata e una gestione attenta del ciclo di vita del software.

Questo capitolo illustra le fasi principali del ciclo di sviluppo dell'applicazione, dalla raccolta dei requisiti iniziali alla distribuzione finale, con particolare attenzione agli strumenti, ai framework utilizzati e alle pratiche di sviluppo adottate e il mio contributo in queste attività.

Inoltre, vengono esaminati i processi di testing, deployment e manutenzione, che sono fondamentali per garantire la qualità, l'affidabilità e la sicurezza del prodotto finale.

4.1.1 Raccolta e analisi dei requisiti

La fase di raccolta e analisi dei requisiti rappresenta il punto di partenza per lo sviluppo dell'applicazione. In questa fase, vengono identificati gli obiettivi principali del progetto e vengono definiti i requisiti funzionali e non funzionali. Il coinvolgimento degli stakeholder, inclusi analisti, manager e responsabili IT, è cruciale per comprendere le necessità operative e le aspettative in termini di funzionalità e performance. Il risultato di questa fase è un documento di requisiti dettagliato che guida tutte le

fasi successive del ciclo di vita. Una delle attività essenziali di cui mi sono occupato, a seguito della raccolta dei requisiti è la realizzazione di documenti in cui vengono proposte le stime, che possono essere di massima o precise, per la realizzazione tecnica dei requisiti presentati.

In questi, vengono presentati il numero di giornate necessarie alla realizzazione di un particolare requisito e la pianificazione dei rilasci di tutti i requisiti trattati.

Successivamente un'ulteriore attività di base, che si effettua per tutte le progettualità, è la scrittura di documentazione a supporto, ogni ambito di sviluppo infatti, dopo la raccolta dei requisiti, viene descritto mediante un documento funzionale (AFU) e un documento tecnico relativo ai dettagli implementativi (ATE), questi saranno revisionati, approvati e aggiornati per ogni nuova evolutiva sull'ambito.

4.2 Architettura target

La progettazione dell'architettura software è una fase cruciale per garantire che l'applicazione sia scalabile, manutenibile e sicura. In questo paragrafo, viene descritta l'architettura scelta, implementata con i framework discussi.

L'architettura target, è basata su microservizi, tale scelta non è stata personale in quanto si tratta di un volere del cliente, ed è anche dettata dai framework di sviluppo utilizzati e da tutto l'ecosistema di sviluppo del cliente.

Come precedentemente descritto, questo approccio consente di suddividere l'applicazione in una serie di servizi indipendenti, ciascuno dedicato a un insieme specifico di funzionalità. Tale modularità favorisce la scalabilità, la manutenibilità e la resilienza del sistema.

La mia attività, per ogni ambito di sviluppo, caratterizzato da una sezione di frontend dedicata, è stata fondata sulla realizzazione, mantenimento e aggiornamento di un microservizio di tipo BE4FE e un microservizio di tipo CORE, insieme anche ad un microservizio BE4FE e CORE per delle funzionalità di visualizzazione ed esportazione dei risultati comuni a tutti gli ambiti di sviluppo.

La comunicazione tra il frontend e il backend avviene principalmente tramite API RESTful. I microservizi BE4FE espongono endpoint che vengono consumati dai moduli Angular, gestendo la logica specifica del canale e invocando i microservizi CORE per eseguire la logica di business.

Il modello dei dati viene popolato e mantenuto aggiornato dalle operatività degli utenti sulla web app e dal flusso di dati descritti nella fase di data ingestion.

La figura 4.1 mostra l'architettura del progetto.

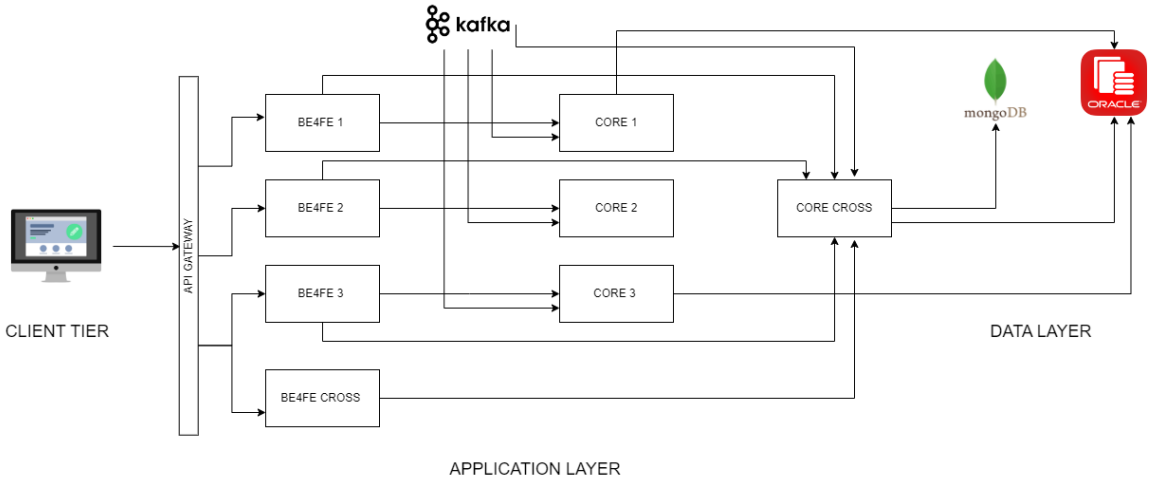

Figura 4.1: Architettura target

4.3 Implementazione

In questa fase, vengono descritti i dettagli tecnici relativi all'implementazione dell'applicazione. Verranno approfonditi i pattern e le best-practices dei framework di backend e frontend utilizzati e il mio apporto personale al progetto.

4.3.1 Sviluppo backend

Lo sviluppo Backend seguirà i pattern del framework del cliente per costruire un'infrastruttura scalabile e modulare basata su microservizi.

Questi microservizi si suddividono principalmente in due categorie: Backend for Frontend (BE4FE) e Core e saranno ognuno per ogni ambito di sviluppo. Questo capitolo esamina in dettaglio lo sviluppo tecnico di entrambi i tipi di microservizi, descrivendo i principali componenti utilizzati e le best practices adottate per garantire un'architettura solida e manutenibile.

4.3.1.1 Componenti principali

I componenti principali da sviluppare sia per il microservizio BE4FE che CORE hanno come scopo quello di orchestrare le REST- Api provenienti dal client e di elaborare eventi nel caso in cui il microservizio preveda l'ascolto di topic Kafka, in modo da gestire la logica di business e trasformare i dati presenti nella base dati per essere interrogati ed analizzati in base alle esigenze degli utenti. Si possono suddividere come di seguito :

- **Controller**, il punto di ingresso del microservizio, responsabile della gestione delle richieste HTTP provenienti nel caso del BE4FE dal client di Frontend, nel caso del core dal BE4FE. il Controller mappa le richieste agli endpoint appropriati e gestisce la validazione iniziale dei dati in ingresso

- **Command**, gestisce la logica di business specifica del microservizio. Implementando il pattern Command, questa componente separa la logica di esecuzione dalla gestione dei dati, facilitando la manutenzione e l'estensione del codice. Il Command può invocare uno o più servizi core per quanto riguarda un BE4FE oppure diversi servizi interni, l'applicazione di regole di business, e la gestione di transazioni complesse nel caso del CORE. Questo componente ha un ruolo chiave nel raccogliere i dati necessari, trasformarli e restituirli al Controller.

- **Service**, componenti che contengono la logica di business riutilizzabile. Nel contesto di un BE4FE, i Service possono essere utilizzati per interagire con i microservizi core, orchestrare chiamate a servizi esterni, o eseguire logiche di trasformazione dei dati. Per quanto riguarda i CORE i service rappresentano il cuore della logica di business, orchestrano chiamate a vari componenti interni e gestiscono operazioni critiche come il calcolo di dati, la gestione di processi aziendali, o l'interazione con database.

- **Connectors**, utilizzati per effettuare chiamate esterne ad altri microservizi o sistemi. Nel caso di un BE4FE, un Connector potrebbe essere utilizzato per invocare API di microservizi core o servizi di terze parti.

4.3.1.2 Design BE4FE

Per lo sviluppo dei microservizi BE4FE (Back-End for Front-End) è fondamentale seguire una serie di linee guida e best practices derivanti dal framework di riferimento, al fine di garantire un'architettura solida, manutenibile e facilmente scalabile. Le pratiche chiave includono:

- **Separazione delle responsabilità**: Il microservizio di tipo BE4fe deve isolare la logica di business dalla gestione delle richieste HTTP, spostando la complessità dell'elaborazione dei dati in componenti dedicati. In questo modo, i controller sono e focalizzati esclusivamente sull'inoltro delle richieste, migliorando la leggibilità e manutenibilità del codice.

- **Validazione dei dati**: La validazione dei dati in ingresso deve essere eseguita nel livello Controller, garantendo che tutte le richieste rispettino i vincoli e requisiti dell'applicazione prima di essere elaborate. Questo approccio previene errori a livello di business logic, riducendo il rischio di elaborazioni non necessarie o errori imprevisti. È possibile sfruttare annotazioni di validazione fornite dal framework, come @Valid e @NotNull, per automatizzare e centralizzare la validazione dei campi.

- **Gestione degli errori**: Per gestire in maniera uniforme e coerente gli errori attraverso tutti gli endpoint, è consigliabile utilizzare eccezioni personalizzate e gestori globali di eccezioni. Questo approccio consente di trattare eccezioni specifiche con messaggi significativi per l'utente, garantendo un flusso di errore ben definito e facilmente tracciabile, senza esporre dettagli interni del sistema. In questo modo si mantiene un'alta affidabilità e si migliora l'esperienza utente.

Queste pratiche contribuiscono a sviluppare microservizi BE4FE flessibili, con una chiara separazione delle responsabilità, facilitando l'implementazione di nuove funzionalità e riducendo il rischio di bug o anomalie nel sistema.

4.3.1.3 Design CORE

In questa sezione si analizzano una serie di caratteristiche fondamentali nello sviluppo dei microservizi CORE, che, come per i BE4FE, devono rispettare specifiche regole architetturali e di progettazione per garantire robustezza e manutenibilità. Le principali linee guida sono:

- **Transazionalità**: È essenziale che le operazioni critiche siano eseguite all'interno di contesti transazionali, per assicurare l'integrità e la consistenza dei dati. Questo garantisce che, in caso di errore, le operazioni vengano annullate (rollback) e non lascino il sistema in uno stato incoerente.

- **Orchestrazione del servizio**: I servizi devono essere responsabili di orchestrare le chiamate a vari componenti o altri microservizi, assicurando che la logica di business sia centralizzata e modulare. La logica deve essere incapsulata all'interno di componenti riutilizzabili, in modo da ridurre la duplicazione del codice e favorire la manutenibilità.

- **Accesso ai dati**: Per gestire l'accesso ai dati, si utilizza il pattern Repository, che consente di astrarre e centralizzare le operazioni di persistenza. Questo approccio migliora la sicurezza e l'efficienza delle operazioni sul database. L'utilizzo di query personalizzate deve essere limitato ai soli casi in cui siano effettivamente necessarie, mantenendo il resto delle operazioni all'interno di metodi predefiniti per garantire maggiore leggibilità e semplicità.

Ognuna di queste regole permette di costruire microservizi robusti, scalabili e facilmente manutenibili nel tempo.

4.3.2 Implementazione dei requisiti

In quanto data engineer, il mio ruolo si è basato principalmente sull'integrazione, sviluppo e manutenzione del backend e quindi dei microservizi BE4FE e CORE degli ambiti di sviluppo.

La richiesta del cliente si concentra su applicazioni resilienti e altamente scalabili vista la mole di dati con cui si lavora. Il mio contributo è stato quello di: analizzare le richieste del cliente per nuove evolutive sugli ambiti di sviluppo, fornire un parere tecnico e sviluppare le soluzioni implementative a determinate richieste. Mi sono anche occupato della manutenzione della web app rispondendo a particolari richieste degli utenti per migliorare la loro esperienza nell'utilizzo della piattaforma.

In questo paragrafo si descrivono una serie di richieste implementative sfidanti e quali sono state le soluzioni tecniche adottate motivando le scelte intraprese.

4.3.2.1 Comunicazione con il legacy

L'ambito di sviluppo che gestisce i fondi ad alto rischio, come descritto nel capitolo precedente, ha un legame molto delicato e complesso con la contabilità, gli utenti svolgono attività su due piattaforme distinte, la prima che alimenta direttamente il database principale con tutte le informazioni di contabilità del legacy e la seconda è la piattaforma descritta in questo progetto.

Quest'ultima consente di eseguire diverse attività periodiche,con cadenza mensile, che si devono riflettere in contabilità avendo cura della validazione dei dati inviati.

Dopo aver attentamente analizzato la richiesta dell'utente, la soluzione implementativa adottata doveva considerare anche il processo di accettazione di nuovi dati in contabilità, il quale avviene mediante ricezione di un file contenente i dati per aggiornare le tabelle del database principale. Di conseguenza, si è scelto il seguente flusso per gestire questa richiesta:

1. La richiesta di nuovi dati da inserire in contabilità avviene dalla piattaforma mediante una chiamata REST ad un particolare endpoint esposto dal BE4FE

2. il BE4FE incapsula la richiesta e la invia al microservizio CORE

3. il microservizio CORE è responsabile di : storicizzare i dati all'interno di una tabella del secondo database e inviare un evento su un topic Kafka in cui è in ascolto il microservizio CORE condiviso dagli ambiti di sviluppo.

4. il secondo microservizio CORE legge il messaggio ricevuto ed inizia la validazione ed elaborazione dei dati, in particolare:

    - Costruisce un file excel, che verrà storicizzato su una collection di MongoDB, in modo tale che gli utenti possano controllare, da una sezione diversa del tool, l'esito della richiesta e nel caso scaricare gli errori che si sono riscontrati
    - Scatena una TWS unattended mediante una REST API

5. Le shell presenti all'interno della tws unattended hanno il compito di costruire il file e di inviarlo verso la contabilità del legacy.

Lo schema di tutto il flusso è rappresentato in figura 4.2.

Il mio contributo è stato quello di implementare ogni fase descritta, come sempre cercando di ridurre a minimo gli errori e le eccezioni che si potessero scatenare e dando massima priorità alla qualità del codice prodotto.

Nella realizzazione e scelta implementativa del flusso sono state analizzate diverse soluzioni, proposte poi al cliente che ha scelto rispetto a particolari bisogni di business e preferenze la soluzione sopra descritta.

Alcune scelte sono "imposte" dal design dell'architettura e dai framework utilizzati. Infatti la richiesta di contabilizzazione che parte dall'utente scatena una chiamata REST che segue il flusso descritto precedentemente, con la realizzazione di un Rest Controller del componente del BE4FE che accettasse una richiesta POST su un particolare endpoint. Successivamente è stato realizzato un componente Command responsabile di incapsulare i dati in input e invocare il relativo Service, quest'ultimo responsabile dell'invocazione dell'endpoint del microservizio CORE su cui è stato realizzato un Rest Controller che accetti le richieste POST provenienti dal BE4FE. Anche in questo caso è stato poi realizzato un Command che incapsuli l'input ricevuto e invochi una serie di Service responsabili di effettuare tutte le fasi a cascata sopra descritte.

In particolare i service del microservizio CORE si relazionano con il database andando ad aggiornare le tabelle di interesse, e/o con altri CORE o servizi esterni.

Una prima soluzione alternativa poteva essere rappresentata dall'utilizzo di un'applicazione schedulata a una particolare ora della giornata che andasse a leggere dalla tabella contenente le informazioni da inviare in contabilità, alimentata dal microservizio CORE, e che fosse responsabile della creazione e invio del file mediante apposite shell. In questo modo il flusso del processo lato Backend sarebbe stato molto più snello e semplice. Questa soluzione però rendeva difficile la gestione della contabilizzazione degli utenti che non sarebbe avvenuta in tempo reale ma al momento dell'orario schedulato dall'applicazione.

La richiesta di esecuzione in real time ha comportato alcune scelte sulla realizzazione del flusso, si è optato per l'invocazione di un secondo CORE mediante l'invio di un messaggio su un topic Kafka su cui questo microservizio è in ascolto. Questa soluzione si implementa mediante l'utilizzo di un particolare Connector messo a disposizione dal framework che espone un metodo che implementato e configurato in base alle proprie esigenze invia il messaggio su un topic Kafka.

Sul secondo microservizio CORE invece, si realizza un listener, grazie sempre a un componente del framework che mediante apposite configurazioni e con l'implementazione di un metodo, ascolta un particolare topic e consuma i messaggi che transitano su esso.

Questo secondo microservizio è responsabile di: costruire un file excel da esporre agli utenti mediante la piattaforma web, così da mostrare loro le informazioni che sono state inviate in contabilità, e richiamare l'applicazione TWS sopra descritta in modo però unattended cioè non schedulata tramite l'utilizzo di un metodo esposto dall'applicazione (trattato nel capitolo precedente) responsabile di invocarla. In alternativa alla costruzione del file excel si potevano mostrare le informazioni inviate in contabilità sulla piattaforma web sotto forma di tabella. Esponendo i dati storicizzati. Questa soluzione però avrebbe "appesantito" l'interfaccia grafica e reso difficile la navigazione della stessa, inoltre il file excel prodotto può essere utilizzato dagli utenti per ulteriori analisi ed elaborazioni.

Un file excel può essere realizzato in diversi modi in un'applicazione Java, si è scelto l'utilizzo della libreria più utilizzata (secondo le statistiche del maven central repository) cioè Apache POI, questa libreria mette a disposizione degli Oggetti che si riflettono sul file excel che si va a costruire, ad esempio il workbook rappresenta l'intero file, lo sheet il foglio di lavoro, row e cell riga e cella del foglio di lavoro eccetera. Si è quindi implementato un metodo che, una volta letto i dati dalla tabella di interesse, costruisse il file excel seguendo un template che era parte del requisito utente.

La conseguente storicizzazione del file costruito poteva essere realizzata in diversi modi. Le alternative erano: l'utilizzo di S3, un particolare contenitore di dati sviluppato da Amazon oppure una collection su un dataBase noSQL in particolare MongoDB. La scelta è ricaduta sull'utilizzo di MongoDB in quanto è integrato meglio con il framework utilizzato ed era già in uso sul progetto, di conseguenza anche per ragioni di omogeneità della progettualità si è scelto tale strumento di storicizzazione. Anche in questo caso è stato realizzato un Connector che questa volta si configura per la connessione a una collection su MongoDB e con cui è possibile definire le query che si vogliono effettuare.

Infine le shell dell'applicazione TWS sono state realizzate in modo conforme alle altre descritte nel capitolo precedente, con la differenza che non eseguono aggiornamenti su tabelle ma ne leggono solo il contenuto per costruire poi un file, mediante

appositi comandi bash, da depositare in una particolare cartella di output del batch-layer. In questo caso, come nei precedenti, si poteva utilizzare un componente Batch scritto con il framework del cliente, che poteva semplificare la costruzione del file mediante la programmazione orientata a oggetti di Java, tuttavia il file da costruire risulta relativamente semplice e non ha bisogno di particolari elaborazioni.

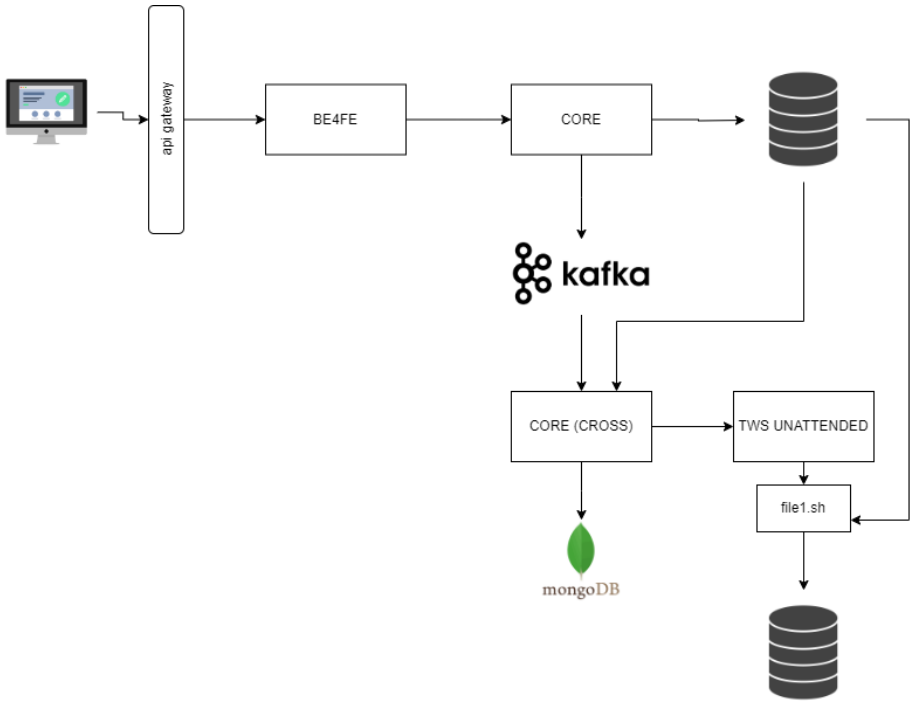

Figura 4.2: Data ingestion del "legacy" mediante piattaforma web

Tutte le scelte intraprese sono focalizzate sul rendere l'esperienza utente più fluida da possibile e incapsulare la logica di invio dei dati in un microservizio che funge da motore centrale di calcolo per tutte le elaborazioni di questo tipo. In particolare l'elaborazione lato utente termina quando il primo microservizio CORE invia il messaggio sulla coda Kafka, a seguito di cui si può continuare con altre attività sulla piattaforma in quanto l'elaborazione si è spostata come detto su un altro microservizio.

Lo svantaggio principale di questo approccio è rappresentato da molteplici punti critici in cui può scatenarsi un'eccezione o un errore e considerando che si tratta di una funzionalità molto delicata è altamente attenzionata e, in caso di errori si ha la priorità massima nella soluzione degli stessi.

4.3.2.2 Esportazione di file di grandi dimensioni

Come descritto in sezioni precedenti, tutti gli ambiti di sviluppo hanno funzionalità comuni gestite da un microservizio separato.

Una di queste è l'esportazione di un numero elevato di estrazioni mediante file excel. Le estrazioni di particolari dati per un determinato ambito di sviluppo, vengono mostrate nella piattaforma sotto forma tabellare e il file excel risultante dall'esportazione deve riflettere la tabella mostrata.

Seguendo un approccio top-down si può suddividere questa richiesta nella soluzione di due problemi:

- Costruzione del file excel
- download del file

Dopo aver realizzato una API, seguendo l'architettura e il design di tutte le API rest del progetto, responsabile di costruire e scaricare il file, si sono analizzate con scrupolosità le performance della piattaforma e e dopo un'attenta analisi, si è giunti a una conclusione implementativa che permette di rendere il più efficiente possibile questo processo.

Si sono suddivise le elaborazioni in due grandi categorie, download diretti e download asincroni, la discriminante tra i due è il numero di righe da dover estrarre, se sono superiori di venti mila, allora il download non potrà avvenire in modo diretto e si dovrà procedere con la costruzione del file in modo asincrono e un successivo download "on-demand", così da non interrompere per lunghi periodi l'attività degli utenti e non sovraccaricare l'elaborazione del microservizio.

Nel caso in cui l'estrazione riguardi un numero minore di venti mila righe allora non ci sono problemi di performance nel costruire il file e permettere il download diretto dello stesso, questa funzionalità è integrata direttamente in HTTP, veicolata da particolari campi presenti nella risposta dell'API.

In caso di download asincrono si è scelto il seguente flusso :

1. L'utente, dopo aver estratto i risultati di una ricerca decide di esportarli mediante apposito tasto mostrato sulla piattaforma
2. Si scatena una chiamata HTTP REST verso un endpoint del microservizio BE4FE che riceve in input tutti i criteri della ricerca effettuata

3. il BE4FE invia la richiesta al microservizio CORE che ha il compito di scrivere su una collection mongo i criteri della ricerca e lo stato dell'estrazione che inizialmente sarà di tipo : "QUEUED" e inviare un evento su un topic Kafka su cui sarà in ascolto il microservizio CORE responsabile delle estrazioni

4. il microservizio CORE responsabile delle estrazioni, legge il messaggio transitato sul topic Kafka e procede con settare lo stato della richiesta in: "TRANSFERING" e con la costruzione del file, mediante l'utilizzo di GridFS. Al termine la richiesta viene settata in stato "COMPLETED" e viene inviata una mail all'utente che aveva chiesto l'esportazione.

5. L'utente responsabile dell'esportazione riceve una mail e può navigare su una sezione apposita della piattaforma per visualizzare la sua richiesta in stato completato e scaricare il file

6. la richiesta di download del file procede come prima mediante una chiamata REST in cui il microservizio CORE scarica il file sempre mediante la gestione di GridFS, ricostruendo i chunk precedentemente costruiti.

Anche in questo caso, il flusso della chiamata REST è conforme al framework utilizzato, la scelta implementativa più impattante risiede nel metodo che costruisce il file, questo rappresenta i dati estratti da un'interrogazione eseguita su particolari tabelle di interesse degli utenti.

Il metodo responsabile della costruzione del file, di conseguenza, deve rieseguire la stessa interrogazione e salvare i risultati prodotti. Le alternative per la realizzazione di questa funzione potevano essere diverse. In particolare si poteva utilizzare un processamento batch on demand mediante invocazione di un'applicazione TWS unattended che eseguisse un componente Batch, responsabile della comunicazione con la tabella in cui è salvata l'interrogazione da eseguire verso il database di interesse, e la successiva costruzione e storicizzazione del file.

Queste operazioni però possono essere svolte da un microservizio CORE ad hoc, configurato ed ottimizzato appositamente per questo tipo di elaborazioni. La scelta è ricaduta proprio su quest'ultima alternativa in quanto rappresenta una soluzione scalabile, mediante opportune configurazioni per gestire il carico sul componente, e trasparente alla piattaforma web che utilizza l'altro microservizio CORE per le sue operazioni.

Preliminarmente, il primo microservizio CORE salva, sulla collection di MongoDB tutte le informazioni dell'esportazione da eseguire tra cui: stato e interrogazione da svolgere. Poteva essere utilizzata anche una tabella di un database Oracle, ma l'utilizzo di MongoDB è essenziale per la successiva costruzione del file, di conseguenza si è scelto di non avere ridondanza dei dati e quindi di salvare fin da subito le informazioni sulla collection di MongoDB.

Viene invocato poi il secondo CORE mediante l'invio di un messaggio su un topic Kafka contente il riferimento della richiesta di esportazione. Questo garantisce asincronicità e resilienza rispetto ad altre possibili implementazioni.

La costruzione del file vero e proprio poi avviene mediante un particolare sistema di gestione dei file di MongoDB cioè GridFS. Anche in questo caso la scelta è ricaduta su questo particolare tipo di database in quanto ben integrato nel framework è conforme al resto del progetto. Si è implementato quindi un Service asincrono su questo microservizio che costruisce i vari chunk del file, legati tra loro mediante un id utilizzato poi per il successivo download.

Il download vero e prorio avviene mendiante una API scatenata dall'apposita sezione della piattaforma web, in cui l'utente vede il nome del file e se questo è disponibile per l'esportazione.

L'API richiamata per il download segue il flusso standard del framework e utilizza l'id dell'esecuzione presente sulla collection MongoDB per ricostruire i chunk del file e inviare come risposta al client il file stesso, che viene scaricato grazie all'utilizzo di particolari Header nella risposta HTTP come Attachment e File-Trasfering.

Poichè rappresentano file di grandi dimensioni, la loro storicizzazione prolungata può comportare un problema infrastrutturale e richiedere un grande quantitativo di memoria. Per questo motivo ad ogni file è associato a una data di scadenza ed è stato realizzato un processo automatico per eliminare tutti i file che avessero superato la loro scadenza.

Questo processo automatico è stato implementato mediante una schedulazione notturna responsabile della funzione sopra descritta.

Anche in questo caso, le scelte architetturali sono state guidate dall'obiettivo di ottimizzare l'esperienza utente, garantendo che essa rimanga il più possibile trasparente rispetto alle operazioni computazionalmente onerose. Allo stesso tempo, si è posta grande attenzione alla resilienza e alla scalabilità del sistema. Nonostante il

flusso preveda l'interazione di numerosi componenti, i punti di fallimento potenziali sono stati minimizzati.

In particolare, le comunicazioni critiche avvengono principalmente con il database, che, per sua natura, introduce raramente errori poichè un eventuale fallimento implica che il database non sia affatto raggiungibile, piuttosto che errori operativi. Inoltre, l'utilizzo di Kafka come sistema di messaggistica garantisce livelli elevati di efficienza e scalabilità, riducendo al minimo le probabilità di errore anche in scenari di carico elevato o distribuzione complessa.

Le figure 5.1 e 4.4, 4.5 rappresentano gli schemi descritti precedentemente in caso di download diretti e asincroni

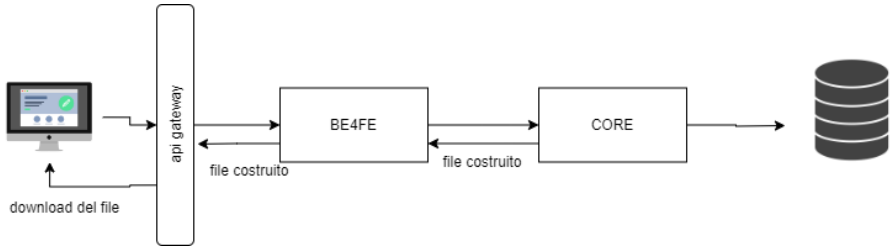

Figura 4.3: download file con meno di 20 mila righe

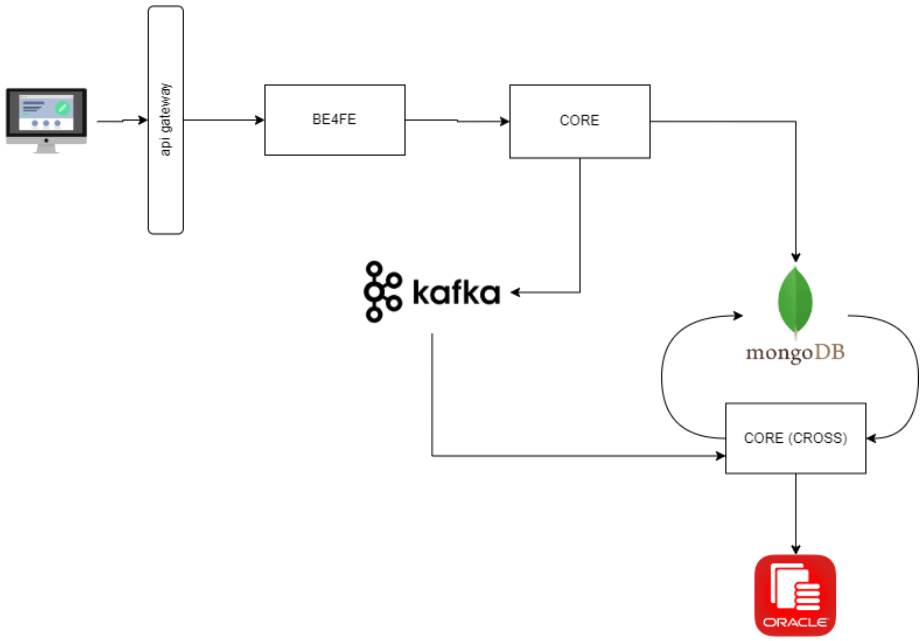

Figura 4.4: costruzione file in modo asincrono

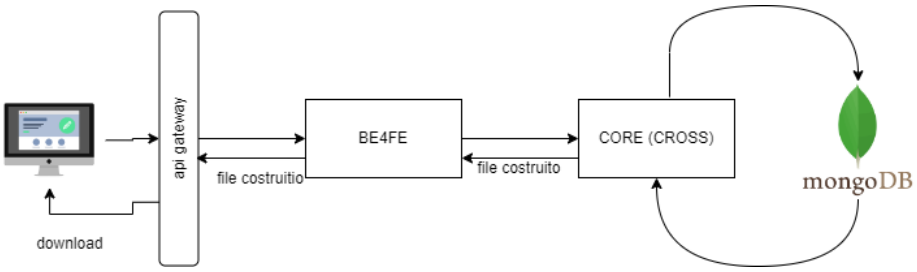

Figura 4.5: download file con un numero maggiore di 20 mila righe

4.3.2.3 Migrazione in cloud

Una delle attività più impegnative che ho svolto, insieme a tutto il team di sviluppo, è stata la migrazione, su piattaforme interamente in cloud, effettuata per tutti gli ambiti di sviluppo per esigenze del cliente.

Per la gestione e orchestrazione dei container su cui sono in esecuzione i microservizi si utilizza OpenShift, in particolare OpenShift Container Platform (OCP), un platform-as-a-service (PAAS) che utilizza Docker come tecnologia per i container e Kubernetes come tecnologia di orchestrazione degli stessi.

L'obiettivo principale del cliente è stato quello di migrare la piattaforma esistente verso OpenShift Container Platform (OCP) versione 4, consentendo il deployment dei vari componenti su infrastrutture cloud offerte da Google Cloud Platform (GCP). In precedenza, l'intero sistema era ospitato su piattaforme on-premise gestite cioè internamente dal cliente.

Con l'aggiornamento a OCP v4 e il passaggio a GCP, il cliente mira a sfruttare appieno i vantaggi del cloud, come la scalabilità dinamica, la maggiore resilienza, la gestione semplificata dei cluster e l'integrazione con i servizi nativi di Google Cloud, superando così le limitazioni imposte dall'infrastruttura locale.

Tale migrazione, per essere correttamente svolta, necessitava di alcuni prerequisiti non rispettati dai microservizi in essere. In particolare le principali modifiche riguardavano l'aggiornamento del framework a una versione più recente e la modifica ai template per la comunicazione con il database e quindi una modifica di tutte le funzioni per la comunicazione verso i database. Queste modifiche hanno comportato le seguenti attività :

- aggiornamento del framework con la modifica delle relative dipendenze

- "refactor" massivo del codice sorgente seguendo le linee guida per l'aggiornamento del framework alla versione necessaria
- No regression testing massivo di tutte le operazioni che si possono eseguire sull'applicazione web negli ambienti di staging non produttivi per assicurarsi di non aver introdotto regressioni.

In particolare, per quanto riguarda l'aggiornamento del framework, si sono utilizzate tutte le note di rilascio dell'ultima versione che è stata implementata, in questo modo si sono capite le differenze rispetto alla precedente e le attualizzazioni da fare lato codice per poter aggiornare il software.

Queste operazioni sono state relativamente semplici, come detto uno dei vantaggi di Spring-Boot, e di conseguenza del framework utilizzato, sono le configurazioni e le dipendenze che mediante file è possibile impostare secondo i propri bisogni.

L'operazione più onerosa è stata la fase successiva all'adeguamento delle dipendenze, quelle aggiornate infatti utilizzano dei metodi con firme, input e output diversi rispetto a quelle utilizzate precedentemente. Di conseguenza è stato eseguito un "refactor" massivo del codice sorgente in modo da adeguare tutte le funzioni che necessitavano di una modifica per via delle nuove dipendenze e che soprattutto continuassero a funzionare allo stesso modo, senza cioè introdurre regressioni. In questa fase quindi, ogni modifica è stata accompagnata da un test che garantisse lo stesso comportamento della funzione.

Particolare attenzione hanno avuto i template che consentono la comunicazione con il database Oracle. In particolare il framework mette a disposizione due tipologie di componenti responsabili della comunicazione con il Database, entrambi molto utilizzati nello sviluppo Java : un template JPA e un template JDBC. Questi due differiscono dalla costruzione della query che si effettua verso il database, JPA infatti sfrutta appieno la programmazione orientata a oggetti di Java e utilizza un particolare linguaggio: JPQL, simile a SQL ma che utilizza il linguaggio a oggetti nelle interrogazioni verso il database. Il JDBC invece necessita che la query venga costruita in linguaggio SQL nativo e all'interno del progetto si fa un uso indifferente dei due template.

Durante la migrazione però le configurazioni del template JDBC hanno suscitato qualche anomalia, con ripetuti timeout durante la connessione verso il database. Questo evidenzia uno svantaggio, descritto nella sezione apposita, dell'utilizzo di Spring-

Boot, complementare alle configurazioni automatiche, queste infatti nascondono tutta la logica sottostante del framework che è di difficile comprensione. Sono state adottate quindi delle configurazioni native del template JDBC, bypassando quelle automatiche imposte dal framework e dopo qualche tentativo, grazie alla collaborazione dei team di supporto apposito, le problematiche sono state risolte.

Dopo aver concluso queste operazioni preliminari, ho contribuito nel processo di migrazione dell'applicativo collaborando durante: le operazioni intraprese dagli altri team coinvolti, le sessioni di testing per assicurarsi il corretto comportamento dell'app e di conseguenza, un monitoraggio attivo in ambiente produttivo durante e dopo il processo di migrazione ultimato.

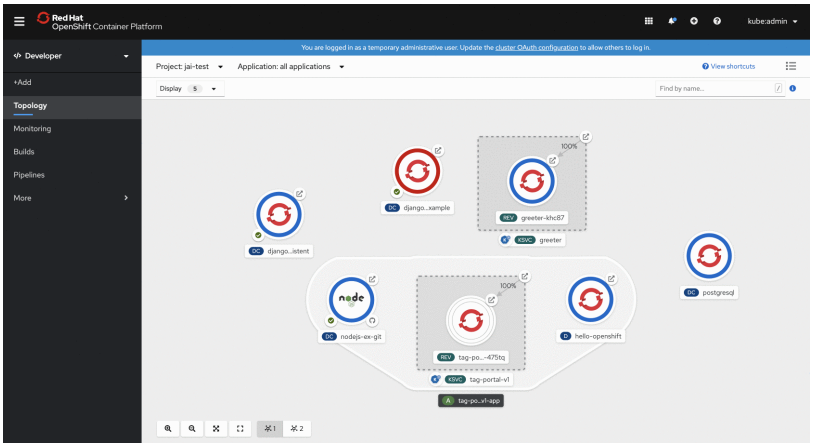

Figura 4.6: esempio di console OpenShift

4.3.2.4 Conclusioni

Lo sviluppo Backend di un'applicazione basata sul framework del cliente richiede una chiara separazione delle responsabilità tra i microservizi BE4FE e Core.

Mentre i BE4FE fungono da intermediari, gestendo le richieste specifiche del frontend e aggregando i dati, i microservizi CORE implementano la logica di business centrale e gestiscono operazioni complesse e critiche. L'adozione di best practices, come l'uso del pattern Command, la gestione delle transazioni, e l'implementazione di repository per l'accesso ai dati, assicura che l'architettura sia scalabile, manutenibile e in grado di rispondere alle esigenze operative dell'organizzazione.

Per quanto riguarda le implementazioni, il framework utilizzato ha concesso tutte le funzionalità richieste con ottime implementazioni delle stesse. L'unica nota dolente

è una difficile personalizzazione delle configurazioni nel caso in cui serva una funzione particolare o complessa, non messa a disposizione o pensata dal framework, anche se questo avviene raramente.

Questo "difetto" però, rappresenta una complicanza nell'utilizzo di un qualsiasi framework sopratutto in quelli strutturati come può essere Spring-Boot che però standardizzano e rendono omogenei i processi di sviluppo di un'app.

4.3.3 Sviluppo frontend

In questo paragrafo, viene descritto lo sviluppo della parte grafica degli ambiti di sviluppo tramite l'adozione del framework sviluppato dal cliente, precedentemente accennato.

Tale framework è costruito su Angular e impiega un'architettura plug-and-play che facilita lo sviluppo e l'integrazione di moduli riutilizzabili. Verranno descritte in dettaglio le operazioni tecniche coinvolte nello sviluppo frontend, esaminando i componenti principali, le best practices e le strategie per garantire un'esperienza utente coerente e performante.

4.3.3.1 Struttura di un progetto

Lo sviluppo frontend segue una struttura modulare, dove ogni funzionalità è suddivisa in moduli indipendenti che possono essere integrati in diverse applicazioni. Ogni modulo è un pacchetto npm versionato, che consente di gestire le dipendenze e le versioni in modo efficiente, le parti fondamentali per la costruzione del progetto sono le seguenti.

- **Moduli**: un modulo in Angular come nel framework usato, è un contenitore di componenti, che possono essere utilizzati insieme per realizzare una parte specifica dell'interfaccia utente.

- **Componenti**: I componenti sono le unità fondamentali dell'interfaccia utente in Angular. Ogni componente è composto da una classe TypeScript, un template HTML e uno stylesheet CSS o SCSS. Il framework proprietario del cliente enfatizza l'importanza della riusabilità dei componenti, che devono essere progettati per essere utilizzati in diversi contesti senza dipendenze specifiche dall'applicazione.

- **Servizi**: I servizi in Angular sono utilizzati per gestire la logica di business, l'accesso ai dati e la comunicazione tra componenti. Il framework del cliente utilizza i servizi per: interagire con le API backend, fornire dati a più componenti e gestire lo stato dell'applicazione.
- **Direttive e pipe**: Le direttive permettono di estendere il comportamento degli elementi HTML, mentre le pipe sono utilizzate per trasformare i dati nelle viste. Nel progetto, queste funzionalità devono essere generiche e configurabili, per essere riutilizzabili in diverse parti dell'applicazione.

4.3.3.2 Gestione delle dipendenze

Ogni modulo è gestito come un pacchetto npm separato, versionato e distribuito attraverso repository privati o pubblici. Questo approccio consente di aggiornare i moduli indipendentemente e di gestire le dipendenze tra moduli in modo efficiente.

4.3.3.3 Implementazione linee guida visive

Il framework utilizzato, impone l'uso di SASS per la gestione degli stili, facilitando l'uso di variabili e funzioni per mantenere la coerenza visiva attraverso l'applicazione. È fondamentale che si utilizzino le classi CSS predefinite fornite dal framework per garantire che i temi possano essere cambiati dinamicamente. Viene supportata anche la tematizzazione dinamica, consentendo di cambiare il tema dell'applicazione in tempo reale senza dover modificare il codice.

4.3.3.4 Integrazione componenti

L'arch-core fornisce componenti e servizi di base che sono utilizzati trasversalmente da tutti i moduli. Questi componenti devono essere integrati nei moduli in modo da sfruttare le funzionalità condivise, mentre le interfacce pluggable permettono personalizzazioni specifiche per ogni applicazione.

4.3.3.5 Conclusione

Lo sviluppo frontend mediante il framework del cliente, implica l'utilizzo di una struttura modulare e scalabile, costruita su Angular e supportata da pratiche avanzate di gestione del codice e DevOps. Non sono presenti particolari sfide nello sviluppo in quanto l'interfaccia grafica è un requisito del cliente che va rispettato ed è resa

omogena su tutta l'applicazione grazie all'utilizzo della libreria di widget messa a disposizione.

Questa tecnologia permette di realizzare applicazioni coerenti e pronte per essere distribuite su diversi canali, tramite l'uso delle best practices indicate, si possono creare interfacce utente che oltre a soddisfare gli standard aziendali, offrono anche un'esperienza utente fluida e intuitiva.

4.4 DevOps

Per garantire che il processo di sviluppo, test e distribuzione sia efficiente, affidabile e scalabile risulta fondamentale la fase di DevOps [10].

Questa fase non è semplicemente un passaggio finale del ciclo di vita del software, ma è integrata in ogni fase dello sviluppo, assicurando che il codice sia costantemente monitorato, testato e pronto per essere rilasciato in produzione. L'approfondimento di queste attività richiedono una comprensione delle componenti chiave e del loro impatto sull'architettura dell'applicazione.

4.4.1 Integrazione continua

L'integrazione continua è una pratica DevOps che prevede l'integrazione frequente del codice sviluppato nei repository condivisi, dove viene automaticamente costruito e testato. Nel caso di questa applicazione ogni commit nel repository attiva una pipeline CI che compila il codice, esegue i test unitari, e verifica che il nuovo codice non introduca regressioni o bug. Questo è particolarmente importante in un'architettura modulare, dove i moduli sono versionati e integrati separatamente. La pipeline CI garantisce che ogni modulo funzioni correttamente sia da solo che quando è combinato con gli altri.

4.4.2 Distribuzione Continua

La distribuzione continua estende la pipeline di integrazione continua per automatizzare la distribuzione del codice su ambienti di staging e produzione:

- **Deployment Automatizzato**: I microservizi, come detto precedentemente, vengono gestiti da Openshift Container Platform che utilizza Docker come tecnologia per i container e Kubernates come tecnologia di orchestrazione degli

stessi. Il versionamento dei moduli di Frontend, permette di aggiornare singoli componenti senza dover ridistribuire l'intera applicazione, riducendo i tempi di disservizio e minimizzando i rischi associati ai deploy.

- **Rollback Sicuro**: Le pipeline CD includono meccanismi di rollback automatico nel caso in cui una nuova release introduca problemi critici. Questo è essenziale in un ambiente dove l'affidabilità e la disponibilità del servizio sono priorità assolute.

4.4.3 Monitoraggio e logging

Un aspetto chiave della fase DevOps è il monitoraggio continuo dell'applicazione e la raccolta dei log per identificare e risolvere rapidamente i problemi, questa attività rappresenta una parte cruciale del mio lavoro quotidiano in particolare:

- Si utilizza Dynatrace per monitorare in tempo reale lo stato dei microservizi e delle componenti frontend, fornendo visibilità su metriche chiave come l'utilizzo delle risorse, il tempo di risposta alle chiamate REST e la disponibilità del servizio.

- I log generati dai microservizi e dai componenti di Frontend vengono aggregati in sistemi di log centralizzati, in particolare si fa uso di Splunk che consente di analizzare rapidamente i problemi, grazie all'esecuzione di query specifiche basate su regex attuate sui log nella loro interezza o su particolari subset definiti nella ricerca. In questo modo ci permette di individuare anomalie e prendere decisioni per la manutenzione e l'ottimizzazione dell'applicazione.

4.4.4 Collaborazione tra team

La fase di DevOps favorisce una stretta collaborazione tra i team di sviluppo e operations:

- **Feedback Continuo**: I feedback sono implementati per garantire che gli sviluppatori ricevano rapidamente informazioni sui test e sullo stato delle release, consentendo correzioni tempestive e miglioramenti continui.

- **Cultura DevOps**: DevOps non è solo un insieme di strumenti e pratiche, ma anche una cultura che promuove la collaborazione, la responsabilità condivisa e l'apprendimento continuo tra tutti i membri del team.

4.4.5 Benefici approccio DevOps

In sintesi, la fase di DevOps è essenziale per garantire che lo sviluppo dell'applicazione sia agile, sicuro e conforme agli standard di qualità aziendali. Integrando pratiche di CI/CD, monitoraggio continuo, sicurezza e collaborazione tra i team, non solo si accelera il time-to-market, ma si migliorano l'affidabilità e la scalabilità dell'applicazione, rendendo possibile un ciclo di sviluppo e distribuzione iterativo e costante.

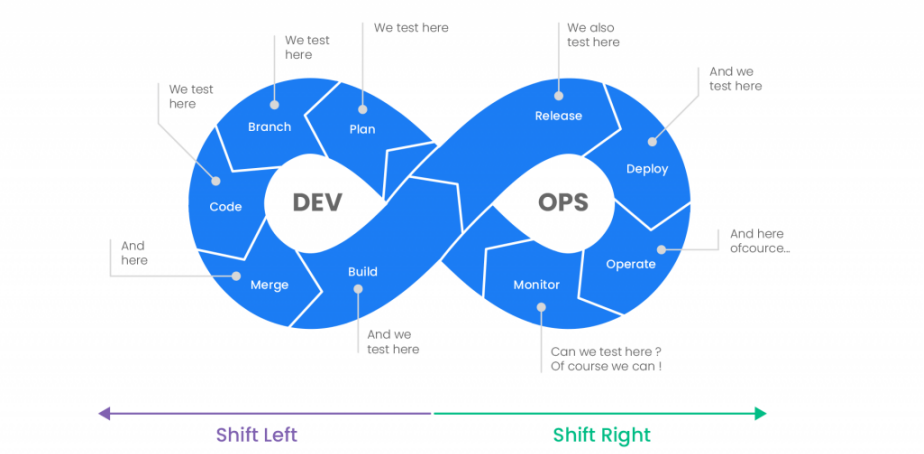

Figura 4.7

4.5 Testing

Il testing è una fase essenziale per garantire la qualità e l'affidabilità dell'applicazione. Si analizzano i diversi livelli di testing adottati, tra cui test unitari, di integrazione e di non regressione descrivendo l'importanza generale dei test e l'uso di framework come JUnit.

4.5.1 Unit Test

Gli Unit Test [11] sono test di basso livello che verificano il corretto funzionamento delle singole unità di codice, come metodi o funzioni, isolandole dal resto dell'applica-

zione. L'obiettivo principale degli unit test è assicurare che ogni componente funzioni esattamente come previsto, indipendentemente dagli altri componenti del sistema.

4.5.1.1 Implementazione e analisi della coverage

Durante lo sviluppo del progetto, il mio contributo ha riguardato anche la creazione e l'implementazione di unit test che rappresentano un elemento fondamentale per garantire la qualità e l'affidabilità del codice sorgente.

Per la realizzazione di unit test, si è utilizzato JUnit, un framework standard che consente la scrittura e l'esecuzione di test unitari in modo efficiente e sistematico. Il processo di creazione dei test in JUnit inizia con la definizione di classi di test, in cui ogni metodo è contrassegnato dall'annotazione specifica, in questo modo si può verificare il comportamento delle singole unità di codice, come metodi e classi, attraverso l'esecuzione di scenari di test mirati.

Ogni test deve soddisfare particolari condizioni che devono essere verificate, denominate asserzioni che confrontano i risultati attesi con quelli effettivamente ottenuti, facilitando l'individuazione di bug e comportamenti anomali.

Per garantire una visione globale della qualità del codice, è stata integrata anche la piattaforma SonarQube, uno strumento potente per l'analisi e la misurazione della qualità del codice sorgente. SonarQube fornisce report dettagliati sulla copertura dei test, evidenziando le parti del codice che non sono state testate. Questa analisi permette di monitorare costantemente la percentuale di copertura, aiutando a identificare le aree che necessitano di ulteriori test e, quindi, a migliorare continuamente la robustezza del software.

Inoltre, SonarQube valuta altri parametri qualitativi, come la complessità del codice e la presenza di bug o vulnerabilità, fornendo un quadro completo della salute del progetto.

Il mio ruolo è stato quello di scrivere test unitari qualitativi e controllare il livello di copertura raggiunta dai microservizi, mediate SonarCube, in modo tale che fosse conforme a quanto richiesto dal cliente e che il progetto non presentasse bug o vulnerabilità.

Questo approccio combinato all'uso di JUnit e SonarQube non solo promuove una cultura di sviluppo orientata alla qualità e alla manutenzione, ma garantisce anche che ogni modifica al codice non comprometta le funzionalità esistenti, assicurando una maggiore stabilità e affidabilità del sistema nel tempo.

4.5.1.2 Vantaggi degli Unit Test

La scrittura ed esecuzione di unit test garantisce i seguenti vantaggi:

- **Isolamento**: Gli unit test permettono di testare singole parti del codice in isolamento, facilitando l'identificazione e la risoluzione rapida dei bug.
- **Velocità di esecuzione**: Essendo test molto specifici e di piccola scala, gli unit test sono generalmente rapidi da eseguire, consentendo un feedback immediato durante il ciclo di sviluppo.
- **Refactoring semplificato**: con una buona copertura di unit test, è possibile effettuare refactoring del codice con la sicurezza che eventuali regressioni verranno individuate tempestivamente.
- **Documentazione vivente**: Gli unit test possono fungere da documentazione esecutiva del comportamento atteso delle unità di codice, rendendo più chiara l'intenzione dello sviluppatore.

4.5.1.3 Svantaggi degli unit test

Di seguito vengono elencati una serie di svantaggi nell'utilizzo degli unit test:

- **Copertura limitata**: Gli unit test verificano solo il comportamento delle singole unità, senza considerare l'integrazione tra componenti. Questo può portare a una falsa sicurezza se l'integrazione tra componenti non è testata adeguatamente.
- **Manutenzione dei test**: Con l'evoluzione del codice, gli unit test possono diventare obsoleti e richiedere aggiornamenti costanti, aumentando il costo di manutenzione.
- **Falsa sensazione di sicurezza**: Un'alta copertura di unit test non garantisce l'assenza di bug, soprattutto quando non vengono considerati i casi d'uso complessi e l'interazione tra componenti.

4.5.2 Integration Test

Gli Integration Test [12] sono progettati per verificare che le diverse unità di codice, una volta integrate, funzionino correttamente insieme, si concentrano sull'interazione tra i moduli dell'applicazione, assicurando che le unità integrate si comportino come previsto nel contesto dell'intero sistema

4.5.2.1 Vantaggi degli integration test

L'adozione di integration test porta con se i seguenti vantaggi:

- **Verifica dell'interazione**: Gli integration test assicurano che i componenti funzionino correttamente insieme, identificando problemi di integrazione che non possono essere rilevati dagli unit test.
- **Rilevazione di bug a livello di sistema**: Poichè gli integration test coprono l'interazione tra più componenti, possono individuare bug complessi legati alla logica del sistema nel suo insieme.
- **Realismo**: Questi test forniscono una visione più realistica del comportamento dell'applicazione, poichè testano le unità di codice nel contesto del loro utilizzo reale.

4.5.2.2 Svantaggi degli integration test

Gli integration test soffrono di una serie di svantaggi, elencati di seguito:

- **Maggiore complessità**: gli integration test sono più complessi da scrivere e mantenere rispetto agli unit test, poichè richiedono la gestione dell'interazione tra più componenti.
- **Tempi di esecuzione più lenti**: poichè gli integration test coprono più unità e possono includere interazioni con sistemi esterni, tendono ad essere più lenti rispetto agli unit test, rallentando il feedback loop.
- **Debugging più difficile**: identificare la causa di un errore negli integration test può essere più difficile, poichè può essere legato all'interazione tra componenti piuttosto che a un singolo modulo.

4.5.3 No Regression Test

I No Regression Test [13] sono test che garantiscono che le modifiche apportate al software non introducano nuove regressioni, cioè che non causino il malfunzionamento di funzionalità precedentemente funzionanti, vengono eseguiti ogni volta che viene introdotta una modifica significativa.

Nel contesto aziendale di questo progetto, il mio ruolo è stato anche quello di collaborare con il team di business per la formazione di un documento formale in cui vengono elencati tutti i test di non regressione da eseguire per un determinato ambito di sviluppo e successivamente procedere con la convalida dello stesso dopo che tutti i test hanno dimostrato un esito positivo.

4.5.3.1 Vantaggi dei No regression Test

Di seguito vengono brevemente descritti i vantaggi dell'esecuzione dei no regression test:

- **Protezione contro le regressioni**: I no regression test forniscono una rete di sicurezza, garantendo che le funzionalità esistenti non vengano compromesse a causa di nuove modifiche.
- **Fiducia nelle modifiche**: La presenza di no regression test consente agli sviluppatori di effettuare cambiamenti con maggiore fiducia, sapendo che eventuali problemi verranno rilevati rapidamente.

4.5.3.2 Complicanze dei No regression Test

L'esecuzione di no regression test provoca alcune complicanze tra cui:

- **Dipendenza dalla qualità dei test precedenti**: l'efficacia dei no regression test dipende dalla qualità e dalla copertura dei test esistenti. Se i test non coprono adeguatamente tutte le funzionalità, le regressioni potrebbero non essere rilevate.
- **Manutenzione e complessità**: con la continua evoluzione del software, la suite di no regression test può diventare grande e complessa, richiedendo significative risorse per la manutenzione e l'aggiornamento.

- **Tempi di esecuzione elevati**: Poichè i no regression test tendono a coprire un'ampia gamma di funzionalità, possono richiedere molto tempo per essere eseguiti, specialmente in applicazioni complesse, rallentando il ciclo di rilascio.

4.5.4 Conclusioni

Gli unit test, gli integration test e i no regression test sono tutti fondamentali per garantire la qualità e l'affidabilità di un'applicazione, ognuno di questi test ha un ruolo specifico nel processo di testing, con i propri vantaggi e svantaggi.

Un approccio di testing completo combina tutti e tre i tipi di test per coprire le diverse sfaccettature del codice e assicurare che l'applicazione non solo funzioni correttamente nel suo insieme, ma sia anche resistente alle modifiche e alle evoluzioni future.

In questo progetto sono state utilizzate tutte le tipologie di test descritte con ulteriore supporto agli utenti finali dell'applicazione durante i test eseguiti da loro stessi per accertare e convalidare ulteriormente il giusto comportamento dell'applicazione.

4.6 Manutenzione ed evoluzione

L'ultima fase del ciclo di vita dell'applicazione è legato alla manutenzione ed evoluzione della stessa, verranno descritte tutte le pratiche di manutenzione correttiva, adattativa e evolutiva necessarie per rispondere ai cambiamenti nel contesto operativo e alle nuove esigenze degli utenti.

4.6.1 Utilizzo di Service Now

Nell'ambito dell'Application Management, la gestione efficace degli incidenti è cruciale per garantire la continuità operativa e minimizzare l'impatto sui servizi offerti, questa è stata una delle attività più importanti da me svolte e che ho seguito durante tutto l'arco di sviluppo del progetto.

ServiceNow [14] è una delle piattaforme più adottate per la gestione IT, fornendo un set completo di strumenti per tracciare, risolvere e prevenire incidenti. Di seguito si descrive l'utilizzo di ServiceNow per la gestione degli incident nell'ambito dell'Application Management, descrivendo le funzionalità chiave, le best practices, e il flusso operativo tipico all'interno di un'organizzazione.

ServiceNow è una piattaforma cloud-based che offre una suite di applicazioni per la gestione dei servizi IT, tra cui l'incident management, consentendo di gestire gli incident in modo centralizzato, facilitando il coordinamento tra i vari team IT e assicurando una risoluzione tempestiva.

Figura 4.8: Servie Now

4.6.1.1 Gestione Incidenti

Quando si verifica un problema in un'applicazione, viene censito un incidente tramite ServiceNow. Questo può essere fatto automaticamente attraverso strumenti di monitoraggio integrati o manualmente da un utente o un operatore di help desk. L'incidente include informazioni dettagliate come la descrizione del problema, la priorità, la categoria, l'utente o il servizio affetto, e lo stato attuale.

Una volta creato, l'incidente viene classificato in base alla sua gravità e urgenza, utilizzando criteri predefiniti per determinare la priorità dell'incidente, che può variare da P1 (critico) a P4 (bassa priorità).

L'incidente viene quindi assegnato al team appropriato per la risoluzione. Una volta assegnato, il team di risoluzione inizia l'indagine dell'incidente utilizzando gli strumenti diagnostici integrati e le informazioni storiche fornite da ServiceNow.

ServiceNow offre una visualizzazione completa del ciclo di vita dell'incidente, permettendo ai tecnici di vedere tutti gli aggiornamenti, le azioni intraprese, e i commenti associati favorendo una diagnosi più rapida e accurata del problema.

Una volta identificata la causa, il team procede alla risoluzione dell'incidente che può includere l'applicazione di patch, la modifica delle configurazioni, o il riavvio dei servizi.

ServiceNow automatizza le comunicazioni relative agli incidenti, inviando notifiche agli utenti interessati e ai responsabili a ogni cambiamento di stato significativo (ad esempio, quando l'incidente viene assegnato, aggiornato o risolto).

Nel caso di questo progetto, sono state definite delle code in cui transitano gli incidenti appartenenti a un determinato ambito di sviluppo. Durante l'apertura dell'incidente il referente applicativo potrà decidere la coda da utilizzare per notificare i relativi tecnici dell'apertura dell'incidente e quindi dell'anomalia riscontrata.

4.6.1.2 Benefici dell'utilizzo di Service Now

L'adozione di ServiceNow per la gestione degli incident nell'ambito dell'Application Management offre numerosi vantaggi:

- **Riduzione dei tempi di risoluzione**: L'automazione e l'integrazione con strumenti di monitoraggio consentono di ridurre drasticamente il tempo necessario per rilevare, diagnosticare e risolvere gli incidenti.
- **Miglioramento della trasparenza e della tracciabilità**: ServiceNow offre una visibilità completa su tutto il ciclo di vita degli incidenti, migliorando la trasparenza e la tracciabilità delle operazioni IT.
- **Ottimizzazione delle risorse**: L'assegnazione automatica degli incidenti e l'utilizzo di una Knowledge Base centralizzata consentono un utilizzo più efficiente delle risorse del team di supporto, riducendo il carico di lavoro manuale e aumentando la produttività.
- **Miglioramento continuo**: Le funzionalità di reportistica e analisi di ServiceNow aiutano a identificare le cause ricorrenti degli incidenti, permettendo all'organizzazione di implementare miglioramenti continui nei processi e nelle tecnologie utilizzate.

In conclusione ServiceNow è uno strumento potente per la gestione degli incident nell'ambito dell'Application Management, offrendo un approccio strutturato e integrato che aiuta le organizzazioni a mantenere l'affidabilità e la disponibilità delle loro applicazioni critiche.

Attraverso l'automazione, l'integrazione e la trasparenza, ServiceNow consente di affrontare e risolvere i problemi in modo attivo, migliorando la qualità del servizio e riducendo l'impatto degli incidenti sui processi aziendali.

4.6.2 Gestione delle evolutive

La gestione delle evolutive rispetto all'AS-IS è un processo cruciale per garantire che l'applicazione possa adattarsi e crescere in risposta alle esigenze aziendali in continua evoluzione.

Questo processo implica l'identificazione, la pianificazione, lo sviluppo, e l'implementazione di nuove funzionalità o miglioramenti all'interno del progetto già convalidato, esistente e funzionante, senza compromettere la sua stabilità o le sue prestazioni.

Nella pratica, ogni modifica proposta passa attraverso un ciclo di vita ben definito che include la valutazione dell'impatto, la progettazione tecnica, lo sviluppo in ambienti di staging, il testing rigoroso, e infine il rilascio in produzione tramite pipeline di Continuous Integration/Continuous Deployment.

Le evolutive vengono gestite in modo iterativo e incrementale, assicurando che l'applicazione possa continuare a fornire valore agli utenti finali, richieste di miglioramento dell'esperienza utente o funzionalità nuove.

Questa gestione delle evolutive offre numerosi benefici che contribuiscono a mantenere l'applicazione rilevante, efficiente e allineata alle esigenze aziendali e degli utenti.

Uno dei principali vantaggi è la continua adattabilità: attraverso l'implementazione di nuove funzionalità e miglioramenti, l'applicazione può evolversi in risposta alle richieste degli utenti, garantendo un'esperienza sempre aggiornata.

Inoltre, questo approccio incrementale riduce significativamente i rischi associati a grandi rilasci, poichè le modifiche vengono implementate e testate in piccoli blocchi, migliorando la stabilità e l'affidabilità dell'applicazione.

La gestione efficace delle evolutive consente anche una migliore allocazione delle risorse, poichè le richieste di nuove funzionalità vengono prioritarizzate in base al loro impatto, ottimizzando il ciclo di sviluppo.

Capitolo 5

Analisi dei risultati ottenuti e sviluppi futuri

5.1 Analisi dei risultati ottenuti

In questo capitolo si analizzano i risultati ottenuti nella realizzazione del progetto in ogni sua parte, approfondendo in particolare la fase di estrazione, trasformazione e caricamento dei dati e la piattaforma web sviluppata con le tecnologie di Backend e Frontend descritte nel capitolo precedente.

Ogni ambito di sviluppo del progetto, è stato realizzato con precisione, garantendo che i requisiti del cliente fossero pienamente soddisfatti.

5.1.1 Analisi dei risultati del processo di ETL

Come descritto precedentemente, le fasi di E.T.L dei dati sono state cruciali per raggiungere gli obiettivi richiesti dal cliente.

Gli sviluppi e le scelte tecnologiche adottate si sono dimostrati strategici ed efficaci per soddisfare i requisiti forniti. La combinazione di strumenti avanzati tra cui Tivoli Workload Scheduler (TWS) per l'orchestrazione dei flussi, shell script per l'automazione delle operazioni, SQL*Plus per l'esecuzione di query SQL, componenti Batch per il caricamento massivo dei dati e IBM DataStage per la gestione delle trasformazioni complesse ha consentito la realizzazione di un processo ETL efficiente e robusto, permettendo di gestire grandi volumi di dati con precisione e tempestività, mantenendo elevati standard di accuratezza e affidabilità.

In particolare, le attività di cui mi sono occupato per quanto riguarda le estrazioni dei dati specificatamente per ambito di sviluppo hanno raggiunto i risultati sperati garantendo affidabilità al processo e correttezza di esecuzione.

I risultati ottenuti sono stati analizzati mediante l'utilizzo di DWC, console descritta precedentemente, che ha permesso un monitoraggio continuo e un'attenta verifica dell'esecuzione di ogni job e relativa shell delle applicazioni schedulate e unattended.

Dalle analisi effettuate è stato possibile perfezionare il processo minimizzando gli errori, questa fase però non si deve mai ritenere conclusa e quotidianamente occorre verificare la corretta esecuzione delle applicazioni fornendo un intervento tempestivo in caso di errori.

5.1.2 Analisi della piattaforma web realizzata

Una fase fondamentale di questo progetto è stata la realizzazione della piattaforma web che mostra i dati, precedentemente estratti e caricati nelle tabelle delle basi di dati apposite, agli utenti finali per le loro attività quotidiane.

Ogni ambito di sviluppo aveva una serie di requisiti da rispettare e funzionalità da sviluppare imposti dal cliente.

Lo strumento finale realizzato, risulta essere funzionale, intuitivo e performante, dimostrando di soddisfare tutti i requisiti del cliente.

Dopo il deploy dei componenti nell'ambiente produttivo, io e tutto il team abbiamo contribuito nel monitorare costantemente la web application per analizzare performance e scalabilità dell'infrastruttura prodotta.

Mediante l'utilizzo di Dynatrace [15] si sono raccolti i primi risultati e metriche prodotte dalla web application. Sulla base di questi risultati, condivisi con il cliente, si sono apportate alcune modifiche per migliorare ulteriormente la performance della piattaforma agendo su particolari interrogazioni al database che risultavano coni di bottiglia per tutta l'applicazione. In alcuni casi, dove vi era la necessità, si è scelta un'implementazione parallela di più interrogazioni critiche verso il database così da fornire una migliore esperienza utente caratterizzata da attese più brevi e una conseguente maggiore reattività dell' applicativo.

Quotidianamente è stata anche eseguita una attenta analisi dei log centralizzati su Splunk, piattaforma che consente di seguire tutto il flusso delle le chiamate REST e la comunicazione tra i microservizi di tipo BE4FE e CORE. Tramite questo strumento è possibile intercettare con tempesitivà gli errori applicativi e procedere nella correzione

degli stessi. Gli errori analizzati sono stati santuari e pochi in quanto il rilascio in produzione è avvenuto successivamente a una fase di testing importante.

In particolare, i risultati ottenuti riflettono una fase di test ben svolta in ognuna delle sue parti. Il mio contributo in questo, come detto, è stato quello di scrivere test unitari che consentissero di raggiungere la coverage richiesta dalle normative aziendali e di collaborare con gli altri team coinvolti e il cliente stesso durante le sessioni di test di integrazione e di non regressione.

Tutte le implementazioni sfidanti, descritte nel capitolo precedente si sono dimostrate efficaci, in particolare dal punto di vista delle performance e della scalabilità. Gli utenti non hanno avuto problemi nell'esportare le loro ricerche in file excel e la comunicazione con il database "legacy" non ha mostrato problematiche.

Inoltre la migrazione alla nuova versione di OpenShift Container Platform con l'integrazione alle piattaforme cloud di Google ha permesso al cliente di eseguire un ulteriore step per la transizione digitale che sta vivendo.

In conclusione, i risultati ottenuti, affermano come i framework per lo sviluppo del Backend e del Frontend proprietari del cliente, siano stati utilizzati e implementati seguendo le best-practices e i loro pattern caratteristici, questo ha permesso di ottenere ottimi risultati garantendo robustezza e scalabilità per le operazioni di Backend e un'esperienza utente fluida e coerente su diversi dispositivi e canali con un'interfaccia grafica che rimane omogonea in ogni ambito di sviluppo, conforme alle linee-guida aziendali.

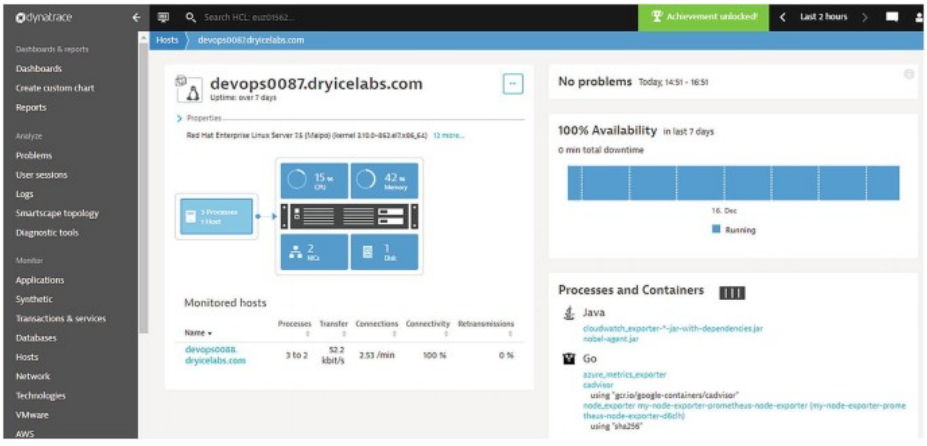

Figura 5.1: esempio console Dynatrace

5.2 Sviluppi Futuri

Sebbene il progetto abbia soddisfatto tutti i requisiti iniziali, è progettato per accogliere futuri sviluppi e nuove funzionalità, garantendo la sua capacità di evolvere in linea con le richieste del cliente.

1. Una delle direzioni principali per gli sviluppi futuri è l'integrazione di nuove funzionalità per la gestione e analisi delle movimentazioni contabili, includendo l'implementazione di moduli di analisi predittiva, che utilizzino algoritmi di machine learning per prevedere tendenze o anomalie nelle movimentazioni contabili, migliorando ulteriormente il supporto decisionale.

2. Il progetto può essere ampliato per includere strumenti di visualizzazione dei dati più sofisticati, come dashboard e report sempre più personalizzati che consentano agli utenti di esplorare i dati in modo più dettagliato e intuitivo.

3. Grazie alla modularità del sistema, il progetto è predisposto per adattarsi facilmente a nuovi requisiti funzionali e tecnici. Questa scalabilità permette di rispondere a nuove esigenze, garantendo che l'infrastruttura sia sempre allineata agli obiettivi strategici dell'organizzazione.

4. Anche se le performance attuali sono ottimali, il monitoraggio continuo tramite strumenti come Dynatrace permetterà di identificare ulteriori aree di miglioramento. L'ottimizzazione delle query SQL, l'adozione di tecnologie di caching e l'affinamento delle pipeline per la data ingestion sono solo alcune delle possibili evoluzioni per mantenere e migliorare le performance dell'applicazione nel tempo.

Capitolo 6

Conclusioni

Tramite questo progetto si é realizzato uno strumento robusto, scalabile e performante, capace di rispondere efficacemente alle esigenze operative e strategiche del cliente.

6.1 Raggiungimento degli obiettivi

Gli obiettivi prefissati, come già analizzato, sono stati pienamente raggiunti.

Il processo di acquisizione, trasformazione e caricamento di grandi volumi di dati, è stato sviluppato utilizzando una combinazione di strumenti e tecnologie che hanno permesso di garantire la qualità e l'integrità delle informazioni.

L'applicazione web sviluppata, permette diverse attivitá agli utenti tra cui: fruire ed elaborare i dati, generare reportistica sofisticata, mediante un'interfaccia intuitiva e performante, capace di soddisfare tutte le richieste del cliente e di essere adattabile nel caso in cui ci fossero nuove esigenze ed evolutive.

6.2 Impatto delle soluzioni adottate

Dopo aver rilasciato l'applicativo in ambienti produttivi, sono stati usati una serie di strumenti come Dynatrace per il monitoraggio delle performance, in questo modo è stata ulteriormente garantita l'affidabilità del sistema.

Le analisi condotte, come descritto precedentemente, hanno evidenziato come l'applicazione sia in grado di gestire carichi elevati senza compromettere le prestazioni, confermando la qualità dell'architettura implementata.

6.3 Contributo al business e all'organizzazione

L'implementazione di questa soluzione ha contribuito significativamente all'efficienza operativa dell'organizzazione, migliorando la gestione delle movimentazioni contabili e fornendo un sistema affidabile per l'accesso e l'analisi dei dati.

La combinazione di tecnologie avanzate e best practices ha portato alla creazione di un'infrastruttura che non solo soddisfa le esigenze attuali, ma è anche pronta a sostenere la crescita e l'innovazione futura del progetto accettando ulteriori requisiti da parte del cliente e nuove esigenze degli utenti.

6.4 Conclusione generale

In conclusione, questo progetto è la testimonianza di come la transizione digitale di grandi imprese sia sempre più al centro dell'attenzione e il conseguente utilizzo ed integrazione di tecnologie avanzate in combinazione a strategie ben pianificate, portino alla realizzazione di soluzioni IT di alta qualità, capaci di fornire un valore reale all'organizzazione.

Tutte le fasi sviluppate per la realizzazione del progetto, non solo rispondono alle sfide attuali, ma rappresentano una base solida per affrontare con successo le sfide future.

Il sistema realizzato è un esempio di come la tecnologia possa essere utilizzata per migliorare i processi aziendali, supportare le decisioni e guidare l'innovazione.

Bibliografia

[1] Advancements and Capabilities in Modern Mainframe Architecture

[2] The What, Why, and How of a Microservices Architecture

[3] Batch Architectural Design Patterns and Tools for Seamless Implementation

[4] Event-driven architecture: cos'è l'architettura guidata dagli eventi

[5] Spring boot

[6] Angular

[7] Spring Batch

[8] IBM Tivoli Workload Scheduler

[9] IBM DataStage

[10] DevOps

[11] What is unit testing?

[12] What is integration testing?

[13] Understanding non-regression testing and its impact

[14] Servie Now

[15] Dynatrace