POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Proposta architetturale a microservizi per il monitoraggio della qualità dei processi di rilascio del software aziendale

Relatore
prof. Giorgio Bruno

Candidato
Juan Sebastian López Moscoso

Anno Accademico 2020-2021

Sommario

L’evoluzione del pensiero umano si riflette nelle sue creazioni. La trasformazione dei sistemi informatici avviene ogni giorno permettendo la scoperta di soluzioni sempre più veloci ed efficaci. Oggigiorno una gran parte dell’economia gira intorno all’evoluzione dell’informatica e porta i prodotti più moderni e adattabili ad essere tra i primi posti tra quelli più venduti. Perciò è importante capire in quale direzione si proiettano questi sistemi e quale sia la soluzione migliore per ogni necessità. Questa tesi si sviluppa intorno all’architettura di microservizi che si presenta come soluzione al problema di un cliente che ha diversi fornitori di informazione e ha bisogno di strumenti che possano far sì che sia l’utilizzo che l’analisi risultino veloci ed efficaci.

Ringraziamenti

Prima di procedere con gli argomenti di questa tesi, vorrei ringraziare a tutti coloro che mi sono stati vicini in questo lungo e difficile percorso di crescita non solo professionale ma anche personale.

Prima di tutto, vorrei ringraziare il mio relatore, il professor Bruno Giorgio per la sua disponibilità e pazienza che mi ha sempre dimostrato. Grazie per trasmettermi le sue conoscenze quando è stato il mio maestro e per essere stato un supporto per questo processo.

Ringrazio l'azienda Sopra Steria che mi ha fornito tutta l'informazione e il supporto necessari per lo sviluppo di questa tesi e continua a farlo nel mio percorso lavorativo.

Ringrazio i miei amici che rappresentano la mia famiglia qui in Italia e sono il mio sostegno principale nei momenti di sconforto.

Grazie alla mia famiglia per non avere mai dubitato di me, per avermi sempre motivato ad andare avanti anche se tutte le circostanze non fossero le migliori, per avermi incoraggiato nei momenti di debolezza e per essere stati forti e fiduciosi nei momenti in cui neanche io pensavo di riuscirci.

Grazie a tutti, senza di voi non ce l'avrei mai fatta.

Indice

Indice delle figure

"La sabiduría nos llega cuando ya no nos sirve de nada"

[GABRIEL GARCÍA MÁRQUEZ]

Capitolo 1 Introduzione

1.1 Introduzione

La realtà in cui viviamo è in continuo cambiamento, così come lo è la tecnologia che ci supporta nelle attività quotidiane. La tecnologia evolve e anche il modo di proiettare le soluzioni si deve adeguare al continuo cambiamento. La necessità crescente di automatizzazione e la gestione delle informazioni attraverso mezzi informatici all'interno delle aziende comportano un grande sforzo di manutenzione ed integrazione. In particolare, nelle grandi aziende i processi di modernizzazione delle infrastrutture avvengono in modo lento e costoso dovuto al cambio delle tecnologie e dei paradigmi soggiacenti ai vecchi sistemi informatici. Si tende ad avere molteplici applicativi che gestiscono indipendentemente le informazioni che richiedono di un notevole sforzo umano per portare a termine i processi di auditing.

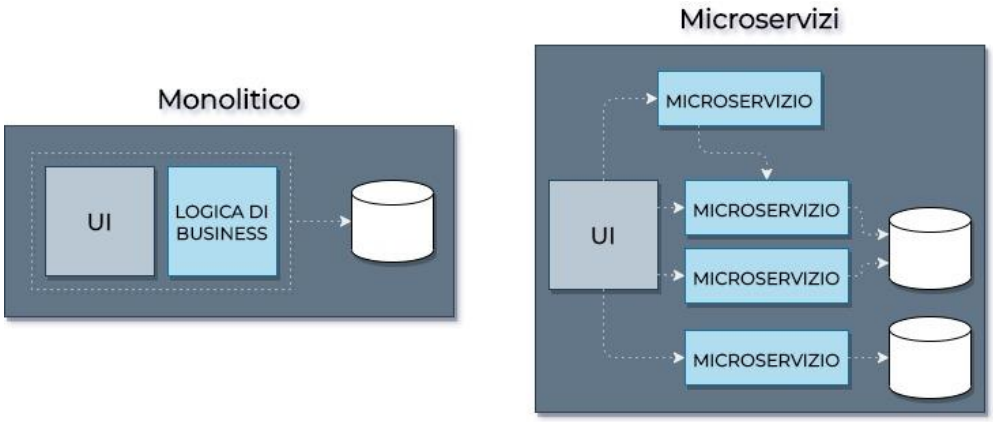

Figura 1.1: Monolitico vs Microservizi.

Con l'evoluzione delle reti e la modernizzazione dei sistemi si introduce il concetto di microservizi che si contrappone alla vecchia architettura monolitica; i sistemi monolitici permettono in modo semplice di costruire un prodotto che si basa su un unico modulo che contiene tutta la logica, mentre i microservizi permettono di frammentare la logica di business in piccoli moduli indipendenti che possono interagire fra di loro, come si mostra nella *1.1*.

1.2 Scopo

Per le grandi aziende avere tanti sistemi monolitici implica un rischio, data la mancanza di flessibilità e la difficoltà di fare interagire i sistemi, soprattutto se i fornitori del software sono diversi. Lo scopo di questa tesi è analizzare, disegnare ed implementare un sistema con delle caratteristiche specifiche di flessibilità, scalabilità, velocità di risposta e semplicità di manutenzione, che dia la possibilità ad una grande azienda nell'ambito bancario di avere un prodotto consolidato per elaborare e mostrare dati provenienti da diverse fonti e metterli in relazione. La proposta è quella di creare una dashboard che permetta di visualizzare diversi widget ad hoc per ogni argomento che mostrano le informazioni più importanti e necessarie per l'analisi degli stati del processo di delivery (messa in azione del software per i processi di creazione, modificazione e manutenzione) dei prodotti software interni all'azienda. La soluzione verrà supportata da servizi che permetteranno di fornire l'informazione ai widget estraendole dai DB predisposti. Questi database verranno popolati in modo continuo da altri servizi (diversi da quelli che espongono i dati), che interagiscono con gli altri sistemi interni dell'azienda per l'acquisizione e l'elaborazione dei dati. In questo modo si otterrà un sistema modulare e flessibile che permette di accentrare l'informazione di monitoraggio del processo d'interesse. Questa architettura modulare consentirà di mantenere i componenti aggiornati indipendentemente dalla tecnologia usata e, in caso di migrazione di piattaforme, di adeguarli in modo semplice ed efficiente.

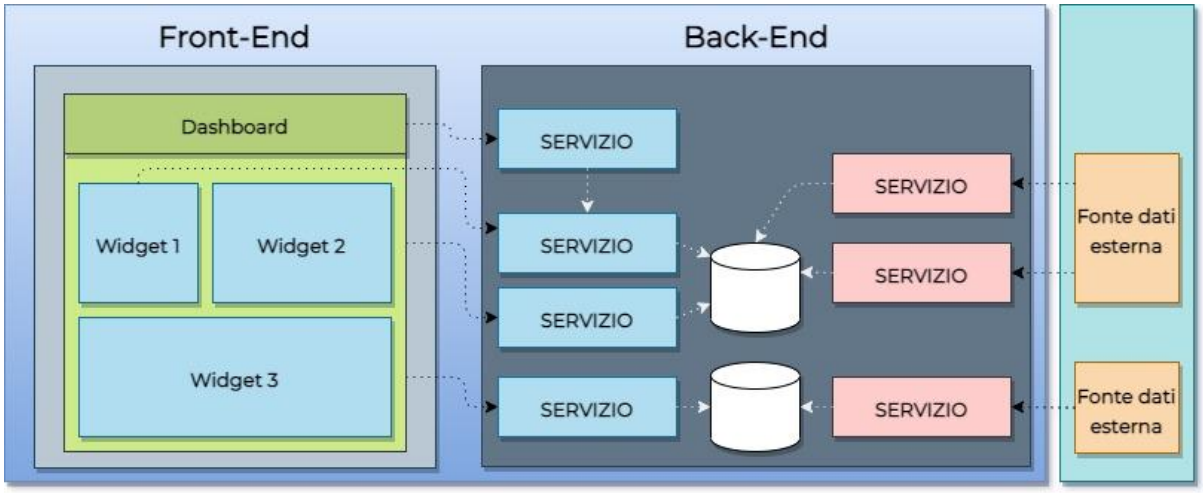

Figura 2: Architettura modulare

1.3 Concetti chiave

L'azienda alla quale si rivolge questo progetto ha già dei sistemi che gestiscono i processi, quindi esistono già dei termini chiave che sono associati a concetti che si svilupperanno nei prossimi capitoli. Inoltre, per la comprensione dell'informazione tecnica fornita è necessario

comprendere la terminologia specifica di tecnologie, di protocolli e di sistemi informatici, per questo motivo di seguito si può trovare un breve glossario dei termini utilizzati:

- ***Progettuali***
    - **QDM (Quality Data Meter)**: è il nome che è stato dato al progetto che si vuole presentare all'azienda ed è il tema centrale di questa tesi.
    - **Progetto**: un progetto è rappresentato da una descrizione e un codice univoco che normalmente ha un suffisso che indica l'origine e la data di progetto.
    - **Step**: ogni progetto viene sviluppato e posteriormente manutenuto, quindi ogni ciclo di sviluppo o manutenzione viene incapsulato in uno step. Ogni step viene rappresentato con una descrizione e un codice univoco che contiene un suffisso che indica il tipo di intervento e a quale progetto appartiene.
    - **Acronimo**: Un acronimo è una sigla che identifica uno o più progetti. Normalmente un acronimo contiene un insieme di progetti che sono legati a livello tematico e che condividono informazioni fra di loro. Questo non vuol dire che la comunicazione con sistemi di acronimi diversi non sia possibile.
    - **AM (Application Maintenance)**: ciclo o gruppo di lavoro che ha come obiettivo fare la manutenzione e le rispettive evolutive agli applicativi. Normalmente i gruppi di manutenzione sono associati solo ad un acronimo per evitare la fuga di informazioni e differenziare il dominio di ogni area.
    - **Rilascio**: ad un processo di rilascio possono essere associati uno o più progetti quindi uno o più step.
    - **Qualità**: la qualità della quale si farà riferimento è quella che può essere calcolata dai diversi parametri ottenuti dai sistemi alimentanti riguardo l'analisi statica del codice, il test planning, la documentazione fornita per la gestione e l'autorizzazione dei diversi cicli di rilascio. In tutti i widget la qualità mostrata riguarderà i dati forniti per uno step specifico.
    - **Change, Tool di change, change console**: Sistema che attualmente gestisce tutto il processo di rilascio e che contiene la maggior parte delle informazioni utili per questo progetto. L'idea non è sostituire il sistema di change, ma integrarlo con risorse visuali che accentrino tutta l'informazione che questo sistema non contiene.
    - **UDC (Unità di cambiamento)**: è il veicolo che permette ai gruppi di lavoro di apportare modifiche ai progetti contenuti in un certo acronimo. Questa unità passa attraverso i diversi ambienti di sviluppo fino ad arrivare in produzione dove il ciclo di vita finisce. Il monitoraggio e la gestione di una UDC avvengono all'interno della change console.
    - **FE (Front-End)**: è un layer ma anche il sottosistema incaricato di presentare in modo grafico i dati utili e completi all'utente con dei criteri estetici definiti e uniformi. Inoltre, deve permettere all'utente delle azioni dispositive per la corretta interazione con i dati. Questo layer deve interagire con il layer di Back-End per permettere tutte le azioni possibili all'utente.

- **BE (Back-End)**: è un layer ma anche il sottosistema incaricato di processare, gestire e modificare i dati in modo logico. In questo layer il sistema interconnette il Front-End e i dati (salvati normalmente nelle Base di Dati) per permettere le diverse funzionalità automatiche o eseguite dall'utente.

- ***Informatici***
  - **Design Patterns**: sono modelli standard che indicano il comportamento di alcun/i elemento/i per incrementare, in modo ottimale, qualche proprietà del sistema. Questi modelli sono orientati a concetti che permettono la scrittura uniforme e sintetica del codice basato sul linguaggio in cui si sta scrivendo.
  - **Publish/Subscribe**: è un design pattern utilizzato nella comunicazione fra componenti asincroni in cui un elemento, chiamato soscrittore o subscriber, attende un segnale di un pubblicatore o publisher per acquisire un messaggio. Un publisher può essere associato a più subscriber e un subscriber può sottoscriversi ai messaggi di diversi publisher.
  - **Application Server**: è un tipo di server orientato ad applicativi web, contenente funzionalità ed infrastrutture per supportare la loro distribuzione (deployment).
  - **Scaling**: è un concetto che può essere inteso come ridimensionamento. Nel software è un attributo di alcuni sistemi che si misura come la capacità di crescita di un sistema a seconda dell'ambiente in cui esiste. Se un sistema è "scalabile", vuol dire che con alcune tecniche di gestione del software, può diventare più grande in termini di gestione dei dati, risorse disponibili agli utenti o qualsiasi area in cui un sistema può diventare più grande.

- ***Tecnologici (linguaggi di programmazione, protocolli, framework, ecc.)***
  - **JavaScript**: linguaggio di programmazione ad oggetti ed eventi comunemente usato per la programmazione web lato client. È un linguaggio che non usa tipi definiti di dati e basa le sue logiche a run-time dipendendo dal contenuto delle variabili.
  - **TypeScript**: linguaggio di programmazione ad oggetti ed eventi, derivato da JavaScript, "strong-typed" quindi con definizione delle tipologie di dati che permettono a compile-time di prevedere casi di errore.
  - **Java**: linguaggio di programmazione ad alto livello, orientato agli oggetti e uno dei più diffusi al mondo.
  - **EventEmitter**: oggetto nel framework Angular che implementa il pattern Publish/Subscribe. Questo oggetto permette ad una classe di fare la sottoscrizione ad un evento attraverso il metodo subscribe() e ad un'altra classe di fare la pubblicazione dell'evento attraverso il metodo emit().
  - **REST (Representational State Transfer)**: è uno standard architetturale che descrive il sistema di trasmissione di dati sul protocollo HTTP. Specificamente,

descrive come devono essere strutturate le URL e i casi in cui si devono usare i metodi descritti dal protocollo http.

- **Json (JavaScript Object Notation)**: è un formato di rappresentazione dei dati utile per il loro scambio fra il client e il server.
- **XML (eXtensible Markup Language)**: è un formato di rappresentazione dei dati basato su “tag” o “markup”.
- **JBoss (Wildfly) Server**: è un tipo di Application Server che mette a disposizione una serie di elementi logici e infrastrutturali per supportare il deployment di applicativi web con certe caratteristiche.
- **Docker**: piattaforma di esecuzione per l’automatizzazione della distribuzione di applicativi software all’interno di contenitori, appoggiandosi nel concetto di virtualizzazione.
- **Bootstrap**: è una libreria front-end che permette l’uso di componenti personalizzabili contenenti logiche di visualizzazione.
- **Quartz**: libreria java che permette lo scheduling di attività con certa periodicità. Il framework Spring presenta un’integrazione dei moduli Quartz che funzionano grazie ad alcune configurazioni specifiche.

Capitolo 2 Requisiti, vincoli e modello architetturale.

L'architettura del software può essere definita in tanti modi dipendendo dall'approccio che si utilizzi. Uno di questi approcci è il processo mediante il quale si analizzano le diverse possibili soluzioni tecnologiche e informatiche per la selezione di un insieme di componenti che permetteranno di soddisfare le necessità di un progetto, oppure, come l'esperto informatico britannico Martin Fowler definisce: "…l'organizzazione fondamentale di un sistema, o il modo in cui i componenti di più alto livello sono collegati fra di loro" (Fowler, Software Architecture Guide, 2019). L'architettura risultante di questo processo cerca di esaudire, il più possibile, le necessità progettuali e di minimizzare le vulnerabilità. Per questi motivi nelle prossime sezioni si metteranno in evidenza i punti principali tenuti in considerazione per la selezione della soluzione più adatta.

2.1 Requisiti utenti

L'utente è un'azienda in ambito finanziario che attraverso gli anni ha costruito un'infrastruttura informatica molto vasta attraverso la concessione di progetti a diversi fornitori. Inizialmente non esisteva uno standard aziendale ben definito per la costruzione delle soluzioni informatiche ma, negli ultimi anni, il processo di standardizzazione e adeguamento del codice ai lineamenti aziendali ha avuto inizio, particolarmente nei nuovi progetti. Il processo di rilascio dell'azienda deve garantire (grazie alle nuove policy) una certa qualità che sarà misurata attraverso la valutazione di diversi indicatori e dovrà essere monitorata e mantenuta attraverso il tempo. Attualmente esistono sistemi diversi e separati che misurano gli indici di qualità, inoltre, ci sono degli operatori che manualmente raccolgono tutte le informazioni dalle interfacce di questi sistemi e monitorano gli indici per procedere con la notifica ai gruppi di lavoro dello stato della qualità progettuale.

Questo è un processo molto costoso in termini di tempo e di risorse umane e, come ogni processo manuale, è soggetto ad errori. Con le nuove policy, il monitoraggio di questi indici diventa critico e si prevede che ci sarà bisogno di implementare un sistema che possa raccogliere e processare l'informazione dei diversi sistemi per calcolare ed evidenziare i punti critici legati ad ogni indice di qualità. L'idea sarebbe quella di trasformare i dati dei sistemi esistenti e dare un'indicazione grafica e compatta agli utenti, per permettere il monitoraggio degli indici della qualità e avere una visione trasversale su tutti i progetti. Quindi il sistema che si vuole creare è indirizzato al gruppo di lavoro dell'ufficio di monitoraggio della qualità del software e dei rilasci.

Per tutto ciò si è pensato di proporre un sistema individuando i seguenti requisiti:

- Il sistema deve permettere di controllare i diversi indici di qualità dei rilasci del software.
- Il sistema deve ottenere ed esporre i valori in un tempo di meno di cinque minuti dalle azioni di rilascio.

- Il sistema deve permettere di compiere un processo di auditing centralizzato (che attualmente viene gestito dal gruppo di quality-release manualmente) con le informazioni dei diversi applicativi.
- Il sistema deve permettere di monitorare lo stato di una unità di cambiamento (UDC) specifica, la documentazione allegata e in generale tutte le policy che l'utente ha disegnato per i ricicli negli ambienti.
- Il sistema deve ricevere i messaggi necessari da parte dei tool di change attraverso delle code.
- Il sistema deve processare i messaggi scodati e rendere disponibile l'informazione necessaria.
- Il sistema deve essere composto da un layer Front-End e un altro Back-End separati.
- Il sistema deve esporre dei servizi che permettano l'invio dei dati al layer di Front-End.
- Il layer di Front-End deve esporre i dati forniti in modo di widget divisi per argomento.
- Il sistema si rende disponibile nella rete interna del cliente e l'accesso è garantito dai server di autenticazione aziendali (fuori perimetro).
- Il sistema deve essere proiettato per favorire la scalabilità e la flessibilità.
- Il sistema deve essere proiettato pensando nelle future implementazioni di ulteriori indici di qualità.
- Il sistema deve poter immagazzinare una grande quantità di dati e deve accedere ad essi in modo efficiente e veloce.
- Funzionalmente devono esporsi quattro diversi widget che organizzino lo stato dei progetti per tematica:
    - Documentazione:
        - Il sistema deve permettere all'utente di selezionare un progetto specifico (codice e descrizione progetto).
        - Il sistema deve permettere all'utente di selezionare lo step specifico del progetto selezionato (codice e descrizione step).
        - Il sistema deve mostrare i documenti e la data di consegna prevista.
        - Il sistema deve mostrare una percentuale di completamento data dal peso e l'obbligatorietà di ogni documento.
        - Il sistema deve mostrare un dettaglio relativo ai dati dello step selezionato e i diversi documenti associati.
    - Indice qualità del rilascio:
        - Il sistema deve permettere all'utente di selezionare un progetto specifico (codice e descrizione progetto).
        - Il sistema deve permettere all'utente di selezionare lo step specifico del progetto selezionato (codice e descrizione step).
        - Il sistema deve mostrare l'indice di qualità del rilascio dato da tutti i parametri acquisiti dai diversi sistemi alimentanti
        - Il sistema deve mostrare i giorni mancanti al rilascio.
        - Il sistema deve mostrare un dettaglio dei responsabili e le date del rilascio.

        - Il sistema deve prevedere un bottone che permetta il blocco del rilascio (si deve implementare un modo di comunicazione fra QDM e il sistema che controlla i rilasci per permettere il blocco).
    - Test Planning:
        - Il sistema deve permettere all'utente di selezionare un progetto specifico (codice e descrizione progetto).
        - Il sistema deve permettere all'utente di selezionare lo step specifico del progetto selezionato (codice e descrizione step).
        - Il sistema deve mostrare i diversi cicli di test.
        - Per ogni ciclo di test, il sistema deve mostrare la quantità di test previsti, il numero di test eseguiti e il numero di test passati.
        - Il sistema deve mostrare il progresso dei test.
        - Il sistema deve poter indirizzare all'utente all'applicativo che attualmente gestisce i flussi di test.
    - Monitor Fabbriche e Qualità del software:
        - Il sistema deve mostrare un calendario che mostri le date e le attività rilevanti per ogni rilascio.
        - Il sistema deve mostrare l'indice di qualità medio del rilascio e gli eventi rilevanti legati alla documentazione.
        - Il sistema deve mostrare il dettaglio dei ticket associati.
        - Il sistema deve mostrare il dettaglio della qualità del software.
        - Il sistema deve mostrare il dettaglio dei costi di AM (a futuro perché si deve prevedere una comunicazione con un sistema che gestisce i costi e i contratti di Application Maintenance).
        - Il sistema deve mostrare il dettaglio degli allegati.

2.2 Processi coinvolti nella raccolta dati

Per capire meglio il legame fra alcuni dei termini usati nei requisiti, sarà necessario spiegare i processi contenuti nei sistemi fornitori di informazione per l'attuale progetto. Il contesto più ampio è stato già descritto e corrisponde all'obiettivo del prodotto che si vuole sviluppare: il sistema di misurazione della qualità dei rilasci.

L'azienda per la quale si sta disegnando questa soluzione prevede un processo di rilascio che avviene tramite una serie di passaggi che permettono di garantire agli utenti degli applicativi una certa qualità. Questo processo parte dalla creazione o manutenzione di un applicativo software, che deve essere testato in diversi modi e approvato da un tester. Dopo tale approvazione si avvia un altro step autorizzativo che passa attraverso il capogruppo e alcuni dei responsabili del progetto che possono approvare o disapprovare il passaggio del rilascio software dipendendo dai valori espressi dagli indici di qualità. A seconda della decisione presa dai responsabili, il processo termina (nel caso di approvazione) oppure viene regredito per il miglioramento di alcune delle caratteristiche del sistema (riciclo).

Ogni passaggio deve essere documentato in diversi file che contengono informazione delle azioni e le modifiche apportate, il modo in cui è stato testato il software, i diversi cambiamenti comportamentali del contesto e dell'applicativo stesso e infine tutti i dettagli che possano essere utili per il rintracciamento di qualsiasi problema che potrebbe generare il rilascio e per consentire, nel caso di cambio di gestore, di dare continuità ai lavori di manutenzione.

Per far sì che tutto il processo si svolga in modo efficace, ci sono delle unità informative chiamate UDC (Unità Di Cambiamento) che permettono il passaggio del software fra i diversi ambienti controllati di sviluppo e testing Le UDC permettono di incapsulare diversi progetti, identificati dai loro acronimi, e per ogni progetto esiste una specie di contatore statico che permette di sapere quante modifiche sono state apportate dalla creazione del progetto. Questa unità viene chiamata Step. Per ogni Step associato ad un progetto possono essere calcolati diversi indici di qualità come quello prodotto dall'analisi statica del codice, il risultato di test manuali eseguiti da persone che usano l'applicativo in uno degli ambienti controllati, la presenza dei documenti nei tempi previsti per la loro consegna e revisione e molti altri processi.

Il controllo dei vari indici è molto importante perché permette non solo di bloccare e sbloccare i processi di rilascio, ma di misurare in modo obiettivo l'andamento dei sistemi e la loro evoluzione attraverso il tempo e i diversi fornitori che le gestiscono. Per fare chiarezza, non tutta la gestione dei sistemi informativi dell'azienda è in mano di fornitori esterni, ma lo è una grande porzione degli sviluppi (soprattutto in ambito tecnico e tecnologico). Alcuni di questi indici esistono già e i loro dati si trovano all'interno di diversi sistemi che, anche se il loro funzionamento interno è ottimale, mostrano solo porzioni di un argomento vasto ed importante come la qualità. In questo modo si rende difficile e lungo il lavoro di monitoraggio su degli indici sia da parte dei responsabili dei progetti, sia da parte del gruppo di quality release monitoring.

2.3 Casi di uso

Per descrivere meglio i servizi offerti dal sistema e le interazione con gli utenti si è disegnato un diagramma di casi di uso. In questo disegno si evidenziano i diversi attori che possono essere gli utenti del sistema QDM e le principali funzionalità offerte. Inoltre, si è deciso di disegnare in modo generico i sistemi esterni nei quali alcuni degli attori forniscono delle informazioni che vengono inoltrate al progetto per fornire dati specifici e concreti.

La parte sinistra della figura evidenzia i sistemi al di fuori del perimetro progettuale e ai diversi attori che effettuano delle azioni rilevanti per lo svolgimento delle attività. La parte destra della figura mostra le componenti principali del sistema che permetteranno la visualizzazione dei dati raggruppati.

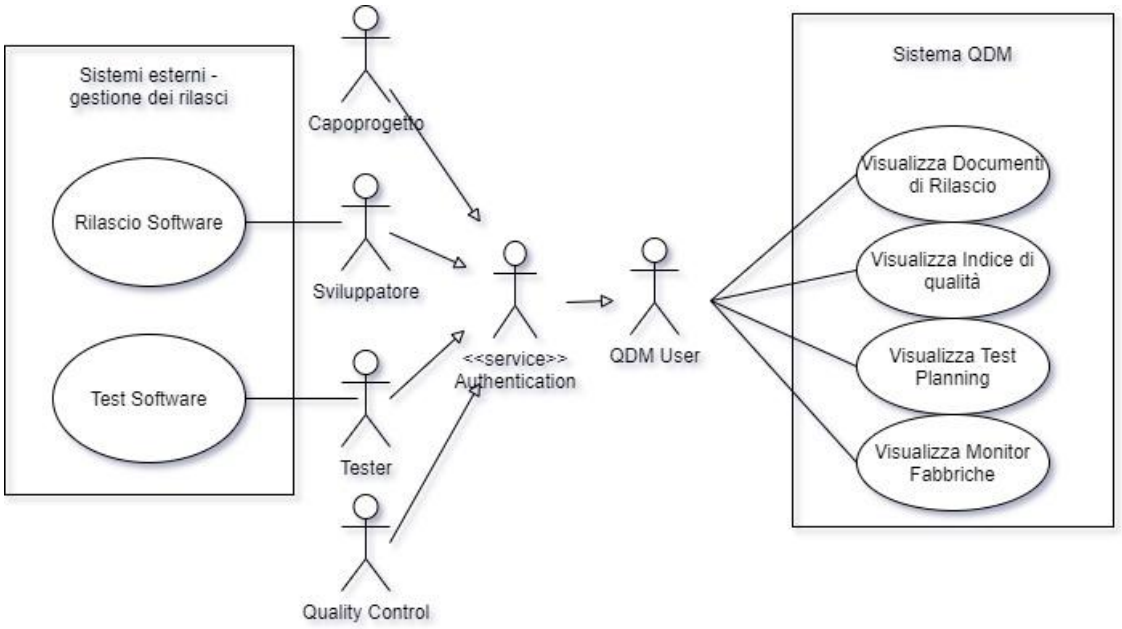

Figura 3: Diagramma casi di uso

Gli attori principali sono:

1. Il capo progetto che è interessato a sapere i diversi indici di qualità dei rilasci all'interno dei suoi progetti. Nell'attualità il capoprogetto non ha nessuna interazione con nessun sistema che gli fornisca informazioni riguardo questo argomento ma l'unico punto di contatto è il gruppo di monitoraggio della qualità dei rilasci software.
2. Quality Control sono tutte le persone che appartengono al gruppo di monitoraggio della qualità dei rilasci software che hanno come compito quello di monitorare i diversi applicativi che gestiscono le informazioni riguardanti la loro area, raccogliere le informazioni rilevanti e analizzarle per fornire un riscontro direttamente ai capi progetto.
3. Lo sviluppatore è l'attore che scrive il codice sia dei progetti nuovi che delle modifiche per manutenzione del software. Questo ruolo ha una visione limitata dell'informazione dato che il suo perimetro si limita alla qualità del software e non a tutti gli altri indici calcolati nelle diverse aree.
4. Il tester è l'attore che fa il test planning e l'esecuzione dei test previsti. Anche il tester ha una visione limitata dell'informazioni degli indici di qualità del rilascio dato che il suo lavoro si centra nell'area di testing manuali e semi-automatici.

Tutti gli attori sopra elencati devono passare tramite un servizio di autenticazione (che è esterno al progetto QDM) diventando così utenti del sistema. All'interno di QDM si trovano tutte le funzionalità di visualizzazione associati agli argomenti rilevanti per lo svolgimento del progetto.

2.4 Vincoli aziendali

Il cliente ha delle limitazioni tecnologiche che non permettono la libera scelta implementativa, dunque la soluzione proposta deve compiere alcuni parametri specifici. Questo avviene perché l'azienda sta passando attraverso un processo di unificazione e standardizzazione dei modelli software sia a livello tecnologico che logico e implementativo, pertanto si stanno generando dei protocolli e aggiornando le policy per il controllo del software rilasciato. Questi sono i principali vincoli e restrizioni che sono stati identificati al momento di presentare il progetto:

- Il software deve essere deployato con la piattaforma Docker.
- Deve essere usata una base di dati relazionale Oracle. In alternativa si può usare una base di dati non relazionale MongoDB.
- I sistemi alimentanti esistono già e si comunicano attraverso file xml dentro delle code che devono essere lette e svuotate nel minor tempo possibile.
- Il sistema deve essere uniforme e facilmente mantenibile.
- La pagina web sarà disponibile soltanto dalla intranet aziendale, quindi, non è necessario implementare un modello di sicurezza.

2.5 Modello architetturale: Monolitico

Quando si affronta il problema della selezione di un modello architetturale si trova davanti ad una decisione importante che impatterà il software in tutto il suo ciclo di vita. Perciò uno degli step più importanti di questo progetto è stata la valutazione di diversi modelli e tecnologie che favorivano certi aspetti ma ne degradavano altri.

In questo caso si sono tenuti in conto due possibili soluzioni a livello architetturale. Il primo è stato un sistema monolitico in cui l'integrazione dei componenti avviene all'interno dello stesso contenitore e non si deve pensare ad un'orchestrazione dei moduli per permettere il corretto funzionamento del sistema. I sistemi monolitici presentano grossi problemi come quello della scalabilità: quando in una grande azienda, un'area cresce in modo diverso da altre, la quantità e la complessità dei dati che si devono gestire diventano importanti e i sistemi devono essere modificati e adattati alle nuove esigenze che la crescita comporta. Pertanto, il sistema monolitico deve essere adattato e deve evolversi in modo da permettere la corretta gestione dell'informazione e garantire che questi cambiamenti non impattino negativamente sul resto del sistema. A questo livello la complessità aumenta in modo esponenziale e i punti critici emergono considerando che si deve evitare a tutti i costi la regressione delle prestazioni e della qualità del sistema in tutte le sue dimensioni. Inoltre, si devono valutare altri aspetti per il quale il sistema monolitico non sembra essere adeguato come soluzione:

1. Il legame ad uno specifico stack tecnologico diventa molto stretto dato che la modifica e, posteriormente, l'adeguamento del codice ad una tecnologia diversa da quella scelta inizialmente diventa molto costoso.

2. Lo scrittore Andrew Hunt e il programmatore David A. Thomas hanno definito l'entropia nel software così: "**Entropia** è un termine della fisica che indica il grado di 'disordine' di un sistema. Purtroppo, le leggi della termodinamica dicono che l'entropia nell'universo tende a crescere. Quando il disordine aumenta nel software, i programmatori parlano di *software rot*, decomposizione del software. " (Thomas & Hunt, 2020).
   Come succede con qualunque sistema, l'entropia nei sistemi complessi e grandi diventa un vero e proprio problema dato che ogni programmatore che mette le mani sul codice sta lasciando la sua impronta specifica data dal suo modo di pensare e sta contribuendo anche inconsciamente alla crescita di questa proprietà o "legge fisica".
3. Qualunque IDE (Integrated Development Environment) usato per apportare modifiche al codice si trova di fronte ad un problema data la grossa quantità di informazioni che sicuramente un sistema monolite contiene.
4. Nel caso una risorsa nuova debba integrarsi al gruppo di sviluppo, deve essere formata non solo con le tecnologie usate all'interno del sistema ma anche con le logiche, gli algoritmi e i diversi pattern seguiti all'interno del codice.
5. Il rilascio applicativo implica che tutto il blocco deve essere riprodotto, validato e consegnato che in termini di tempo non è mai banale.

Per queste grosse problematiche e date le proprietà ricercate nel sistema da sviluppare, si è pensato di valutare un'opzione più flessibile, scalabile e mantenibile a futuro. Queste caratteristiche fanno pensare direttamente ad un'architettura a microservizi.

2.6 Selezione della soluzione: Microservizi

L'architettura a microservizi è il nome dato alla progettazione architetturale di un software composto da moduli staccati che producono risultati "atomici". Ciò vuol dire che ogni segmento logico può essere incapsulato in un modulo che ha una funzione determinata e contribuisce al raggiungimento dell'obiettivo del sistema completo: processare dati e fornire l'informazione completa e dettagliata all'utente. La progettazione modulare permette di rompere il vincolo creato con le tecnologie, le librerie e i framework che si usano al loro interno dato che ogni modulo può essere progettato in modo separato dal resto ed evolvere in modo indipendente riducendo l'eccentricità.

Questo modello, supportato dalla scelta di un framework e delle norme di implementazione, fornisce le caratteristiche che coprono la maggior parte delle necessità di flessibilità che vengono cercate e che sono un peso molto grosso attualmente per l'azienda. L'accoppiamento dei dati e l'accentramento di essi diventa un compito facilmente scalabile dato che questo tipo di architettura permette l'integrazione di nuove logiche nel sistema in modo atomico e quasi indipendente (l'unica dipendenza è a livello logico per non causare problemi di integrità dei dati).

A livello tecnico, lo sviluppo dell'applicativo sarà diviso in due macroaree: il layer di Front-End e il layer di Back-End. Il primo è associato ai componenti grafici, alle logiche legate alla

proiezione dei componenti visuali per dare all'utente un interfaccia amichevole e semplice. Il secondo è diviso in due aree, una per l'acquisizione, processamento e filtraggio dei dati (Engine) e una che permetta la gestione e il passaggio dei dati verso il layer Front-End (Services). Ogni modulo che compone il layer di Back-End deve essere incapsulato in un microservizio che, per disposizioni aziendali, deve funzionare in un container Docker contenente l'immagine di un Application Server JBoss (Wildfly). A livello locale, tutte queste disposizioni sono state rispettate e configurate per la simulazione del sistema.

Il sistema avrà come Base di Dati un Oracle DB: questo data base è una disposizione aziendale che è stata replicata a livello locale come un container Docker contenente l'immagine dell'Oracle DB con le tabelle necessarie per il funzionamento del sistema.

La comunicazione con i sistemi esterni si effettua tramite delle code e dei processi che scodano i messaggi che sono fuori perimetro di questo progetto. Invece l'acquisizione dei messaggi contenuti in file XML e la loro elaborazione e cancellazione dipendono dal progetto Engine che può essere replicato in più container a seconda della necessità di elaborazione.

Si è previsto lo sviluppo di un componente aggiuntivo funzionante come libreria (Core) che contenga funzionalità trasversali e modelli centralizzati per evitare la replicazione e la ridondanza dei modelli come quelli di Entità che modellano gli elementi presenti sul DB.

Tutti questi concetti si possono illustrare nel seguente diagramma:

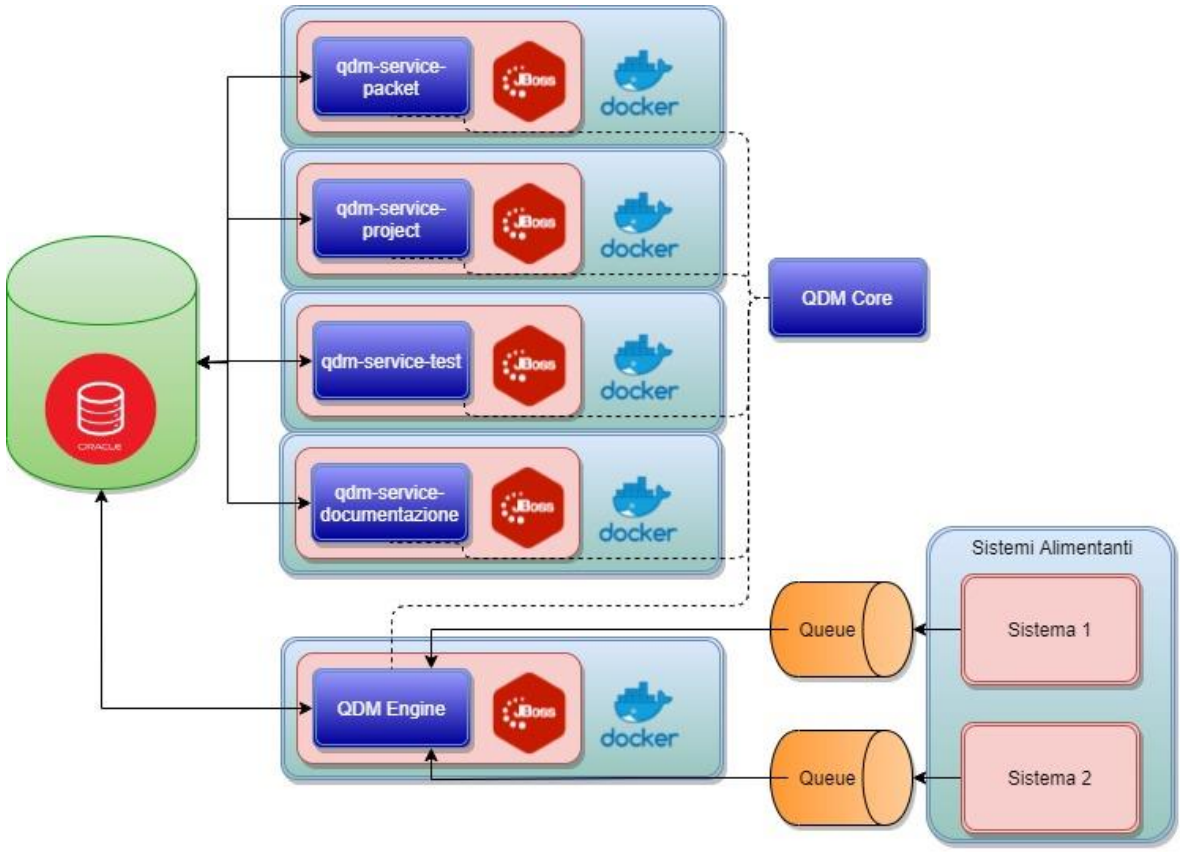

Figura 4: Modello architetturale del BE

In questo modello si può vedere il flusso dell'informazione seguendo le frecce. Invece le dipendenze sono date dalle linee tratteggiate. Si ricorda che le code (illustrate in arancione col nome Queue) e i sistemi alimentanti sono fuori dal perimetro del sistema QDM e vengono esposti solo a scopo illustrativo.

La selezione dei framework e delle tecnologie verrà descritta e specificata nei prossimi capitoli.

Capitolo 3 Implementazione prototipo funzionale

Quality Data Meter, come accennato nei precedenti capitoli, è stato un progetto sviluppato velocemente per dare l'idea al cliente di come la pagina web dovrebbe comportarsi a fronte dei diversi requisiti estratti. Questo progetto è stato implementato a modo di prototipo funzionale nel quale si trovava un'interfaccia grafica che mostrava i dati finti, senza un vero e proprio sviluppo del layer di Back-End.

Il progetto inizialmente prevedeva quattro fasi di sviluppo della proposta (precedenti allo sviluppo vero e proprio dell'applicativo nel caso in cui fosse stata accettata la proposta): fase di analisi e disegno del prodotto, fase di creazione del prototipo funzionale, fase di estensione del prototipo funzionale con l'aggiunta di servizi Back-End, fase finale di integrazione e consolidamento dei requisiti per dare inizio allo sviluppo reale del progetto. In particolare, sono stato coinvolto in tutti i cicli che sono stati effettuati e, dall'abbandono del progetto da parte dell'azienda proponente, è stato compito mio continuare con gli sviluppi in modo autonomo. È necessario chiarire che ho avuto l'autorizzazione per usare l'idea e gli sviluppi effettuati come base per l'elaborazione di questa tesi togliendo soltanto i dettagli che potrebbero generare problemi di privacy.

3.1 Fase di analisi e disegno del prodotto

Durante la fase iniziale è stato necessario interfacciarsi con le persone del gruppo di monitoraggio della qualità dei rilasci, interpretare i flussi di dati e la loro provenienza per capire il contesto del progetto. La necessità di questo sistema era già stata analizzata da alcune persone che lavoravano nel gruppo interessato ma non era mai stata approfondita da nessun gruppo di sviluppo per motivi esterni a questa tesi. Sono stato coinvolto solo nel periodo finale di questa fase, dove in modo discorsivo e informale, sono stati comunicati i requisiti per decidere i linguaggi di programmazione, le tecnologie e i framework da usare. In aggiunta, a livello architetturale si è replicata l'idea di un altro progetto della stessa azienda, adattandola alle necessità specifiche di QDM. Il progetto di cui si è replicata l'idea architetturale è stato selezionato anche come riferimento per la creazione del framework proprio dell'azienda che era sviluppato in contemporanea all'attuale progetto.

3.2 Fase di creazione del prototipo funzionale

La seconda fase prevedeva la creazione del prototipo funzionale. I principali framework scelti sono stati Angular per il Front-End, Spring per il Back-End e la scelta della base di dati è stata Mongo DB.

Il prodotto creato doveva essere continuamente presentato al responsabile dell'area di monitoraggio della qualità dei rilasci che, ad ogni riciclo, forniva un riscontro obiettivo per la creazione di un software adatto alle necessità dell'utente finale. Il prototipo iniziale era

composto da una dashboard principale che mostrava i diversi widget relativi alle quattro aree richieste e una pagina secondaria contenente grafiche delle statistiche che si potevano associare ai dati. Ogni widget si concentrava in un indice di qualità principale che veniva misurato a seconda delle logiche molto banali per dimostrare, solo a scopo illustrativo, le dinamiche che si potevano integrare nel progetto. In questa fase era stata prevista non solo la visualizzazione dei dati ma anche l'interazione con alcuni dei sistemi alimentanti per la loro gestione e modifica. Quindi, l'utente di QDM aveva il potere di compiere azioni come il blocco/sblocco di un rilascio, aggiunta di documentazione e di test.

In questa fase si sono abbozzati i progetti che componevano il Back-End:

- QDM-Core: libreria scritta come supporto centralizzato per gli altri due moduli. Contiene funzioni utilitarie statiche e il modello dei dati persistiti. Inoltre, contiene i repositori che implementano le query più frequentemente usate per togliere carico ai moduli progettuali.
- QDM-Engine: modulo contenente le logiche di traduzione (identificato dal nome in inglese "Parsing") dei messaggi derivati dai sistemi esterni e il rispettivo salvataggio nella base di dati. Per ogni sistema alimentante si è previsto almeno un tipo di file XML specifico per descrivere i dati che dovevano essere letti, trasformati, processati e salvati. L'intestazione di ogni file permette al modulo Engine di capire quale dei Parser deve essere usato come traduttore dell'informazione contenuta. L'espansione di questi parser è avvenuta in modo veloce e facilmente adattabile. Il modulo rappresenta solo un modo di conversione dei dati perché non è stata data l'autorizzazione per l'acquisizione dei file interni ai flussi reali. Quindi, il modulo permette di lavorare dati finti ma verosimili.
- QDM-Services: modulo corrispondente all'unico microservizio originalmente disegnato. Questa componente conteneva tutte le funzionalità per l'esposizione dei dati e per la loro gestione. I componenti interni sono stati divisi secondo i quattro argomenti principali e la loro rappresentazione nella base di dati, così da a vere un isolamento logico.

Personalmente, in questa fase avevo la responsabilità degli sviluppi del Front-End con il supporto di una figura che funzionalmente conosceva una gran parte dell'infrastruttura ed era in grado di fornire dati verosimili per mostrare nelle pagine previste (dashboard e statistiche). Per questioni di licenze e di policy aziendali del cliente, non era possibile usare molti dei componenti web provenienti da fonti esterne (librerie), dunque è stata necessaria la costruzione della maggior parte dei componenti grafici usati.

3.3 Fase di estensione del prototipo funzionale: aggiunta del layer di Back-End

La terza fase non è stata svolta completamente perché, mentre gli sviluppi del back-end cominciavano, la proposta è stata rifiutata da uno dei capoufficio e quindi le persone allocate a questo progetto sono state assegnate ad altri progetti in corso. Uno dei motivi principali per i cui è stato rifiutato il progetto è stata la sovrapposizione dei perimetri di azione con i sistemi alimentanti, dato che agli utenti dell'applicativo venivano concessi certi poteri di gestione dei rilasci e le aree che lo compongono. Questo implicava di dover adeguare tutti i sistemi per l'interazione con QDM e, per motivi di costi e di complessità aggiuntiva negli applicativi esistenti, non è stato possibile procedere con il progetto.

Il sistema è rimasto parzialmente sviluppato; il progetto Engine è stato quasi portato a termine così come il modulo Core. Le funzionalità previste per l'invio dati verso il Front-End sono state abbozzate. Inoltre, i widget disegnati per la dashboard del prototipo funzionale sono stati riutilizzati ed erano in processo di adeguamento, con l'obiettivo di mostrare l'informazione proveniente dai servizi Back-End.

Con lo stato del sistema in quel momento, ho dovuto completare le attività di scrittura dei servizi lato Back-End e gestire i flussi dati all'interno del progetto Front-End per popolare i widget. Ma le qualità del progetto non sodisfacevano i requisiti di flessibilità e scalabilità pretesi, neppure il modello corrispondeva ad un disegno architetturale a microservizi, quindi le modifiche più importanti sono state le seguenti:

1. Adeguamento architetturale basato su quattro microservizi organizzati per argomento (stesso incapsulamento che era stato fatto all'interno del microservizio QDM-Services), interfacciati con il front end per il popolamento dei dati nei widget.
2. Adeguamento di tutti i moduli Back-End per passare da un Application Server Tomcat a container Docker con immagine JBoss per il deploy di ogni microservizio. La selezione di Docker è stata fatta per permettere il deploy contemporaneo dei diversi container e fare una simulazione del modello più verosimile e facilmente gestibile.
3. Modifica del tipo di base di dati. La decisione di modificare la base di dati è stata presa dato che un DB non relazionale non era affatto congruente con i dati che dovevano gestirsi e le loro relazioni che avrebbero aggiunto complessità innecessaria nel modello dati. Quindi si è passato da avere un MongoDB ad avere un Oracle DB. Si è disegnato un modello di entità e relazioni che permettesse la gestione e l'acquisizione dei dati in modo semplice ed efficiente.
4. Adeguamento delle funzionalità dispositive (funzionalità che permettono la gestione e modifica dei dati o processi). Tutte le funzionalità che permettevano agli utenti di realizzare qualche azione corrispondente ad un sistema esterno sono state sostituite con bottoni di reindirizzamento agli applicativi esterni che attualmente le gestiscono per evitare precisamente la sovrapposizione dei perimetri di azione.

5. Adeguamento generale dei dati e dei modelli per motivi di privacy. Sono stati cambiati tutti i nomi e i dati forniti nei flussi per adeguare il sistema e la base di dati, con l'obiettivo di evitare il riconoscimento dell'azienda e i suoi componenti interni. Rimangono solo, in modo parziale, alcuni concetti che sono abbastanza generici da non essere riconoscibili. A livello visivo, sono state sostituiti il logo, il nome dell'azienda e tutta la palette di colori.

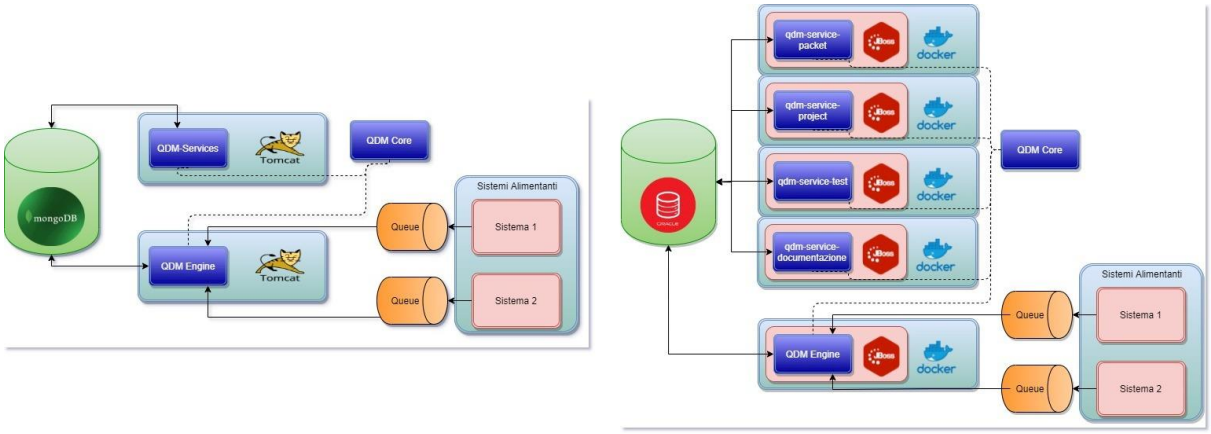

Figura 5: Modifiche architetturali

Tutte le modifiche portano il progetto ad avere delle caratteristiche più adeguate ai requisiti sollevati e ad avere un perimetro ben definito che non interferisca con i sistemi circondanti. Quindi QDM nella versione attuale diventa un sistema di monitoraggio dei diversi indici di qualità dei rilasci software, che permette al gruppo di quality release monitoring di intervenire tempestivamente nei casi critici attraverso i meccanismi previsti e di storicizzare i dati per permettere l'analisi dell'evoluzione dei sistemi software.

I dettagli implementativi saranno argomento dei capitoli successivi.

3.4 Fase di integrazione e consolidamento dei requisiti

Evidentemente questa fase non è stata avviata per la cancellazione del progetto. L'unica attività relativa a questa fase è stata di mia iniziativa, perché mi sono permesso di condividere il risultato degli sviluppi con il gruppo di lavoro che hanno approvato le funzionalità e le nuove implementazioni. L'evoluzione e l'adeguamento generale del progetto e stato motivo di crescita sia a livello di conoscenza che professionale.

Capitolo 4 Implementazione del Front-End

L'interfaccia grafica e la UX (User eXperience) sono una parte molto importante dei progetti dato che è quello che permette agli utente di interagire e vedere i dati che sono sotto il dominio applicativo. In particolare, nel progetto QDM, il front-end non fa parte delle attività principali dell'applicativo per motivi contrattuali, quindi l'elaborazione di questo segmento è a scopo unicamente illustrativo. Per la presentazione del progetto si è creato un "Mockup" (modello interattivo con dati fittizi che permette di illustrare i dati e l'usabilità prevista per l'applicativo) che si è mantenuto ma, in un segmento a parte, si è esteso ulteriormente per l'interazione con i servizi esposti dal BE.

4.1 Framework: Angular

La selezione del framework ha dovuto rispettare gli standard e le policy aziendali quindi l'implementazione del Front-End è stata sviluppata col Framework Angular. Il modo migliore per capire che cos'è Angular è citando la descrizione che gli autori di Google forniscono nella loro pagina web: "Angular è una piattaforma di sviluppo […] che include:

- Un framework basato in componenti che permette di costruire applicazioni web scalabili.
- Una collezione di librerie integrate che coprono una grande quantità di funzionalità come il routing, forms, gestione, comunicazione client-server e molto altro.
- un insieme di strumenti per gli sviluppatori che aiutano a sviluppare, costruire, testare e aggiornare il codice." (Google, s.d.)

Come afferma la descrizione, Angular è un framework che favorisce la scalabilità e garantisce elevate performance. Questa piattaforma permette agli sviluppatori di creare dei componenti riutilizzabili e facilmente integrabili che, attraverso meccanismi interni, permettono l'interazione e il flusso dei dati. Parte importante di questa piattaforma è il concetto architetturale di Single Page Application: Angular permette agli sviluppatori di creare una pagina unica che aggiorna i suoi componenti in modo dinamico e consente di effettuare un routing fra diversi componenti senza eseguire un ricaricamento completo della pagina ma solo del suo contenuto. Questo permette agli applicativi web lato client di caricare solo i componenti necessari e, dinamicamente, navigare attraverso le diverse funzionalità e informazioni previste. È anche previsto il caricamento statico delle componenti in cui ci sono dei server statici che provvedono le informazioni necessarie soprattutto a livello di formattazione (CSS) e logico (Librerie JavaScript). In questo modo le richieste al server contenente l'applicativo sono minori e il carico si distribuisce.

Di seguito si descrivono i concetti base del framework Angular che saranno utili per capire lo sviluppo della soluzione.

4.1.1 TypeScript

Angular, a differenza del suo predecessore AngularJS, usa il linguaggio TypeScript (TS) che è basato su JavaScript (JS) ma contiene molte più funzionalità e definisce anche tipologie di dati e contenitori che su JavaScript non esistono. Questo linguaggio permette ai programmatori di implementare e gestire le variabili e gli errori che possono sorgere nelle diverse funzionalità in modo più essenziale ed efficiente. Il codice scritto in TypeScript viene tradotto in codice JavaScript così da essere eseguito nei tool che lo supportano già. In questo modo il programmatore deve gestire i dati in modo più semplice e specifico evitando sia di dover scrivere logiche molto grosse che certi errori dovuti alla mancanza di tipi.

4.1.2 Moduli

Gli NgModules o moduli Angular, sono classi che definiscono una serie di funzionalità ed elementi che sono collegati tra di loro. Questi moduli permettono di organizzare il codice e le funzionalità in modo logica e, infatti, fornire al framework la caratteristica di "modularità". I moduli possono importare ed esportare delle funzionalità così da permettere lo scambio con altri moduli dello stesso applicativo. I moduli sono facilmente riconoscibili grazie al decorator *@NgModule()* che riceve come input un oggetto contenente metadati utili alla descrizione del modulo come "*imports*", "*exports*", "*declarations*", "*bootstrap*" e molti altri.

I moduli provvedono un contesto di compilazione per i componenti al suo interno. Quindi, una applicazione Angular contiene almeno un modulo con minimo un componente.

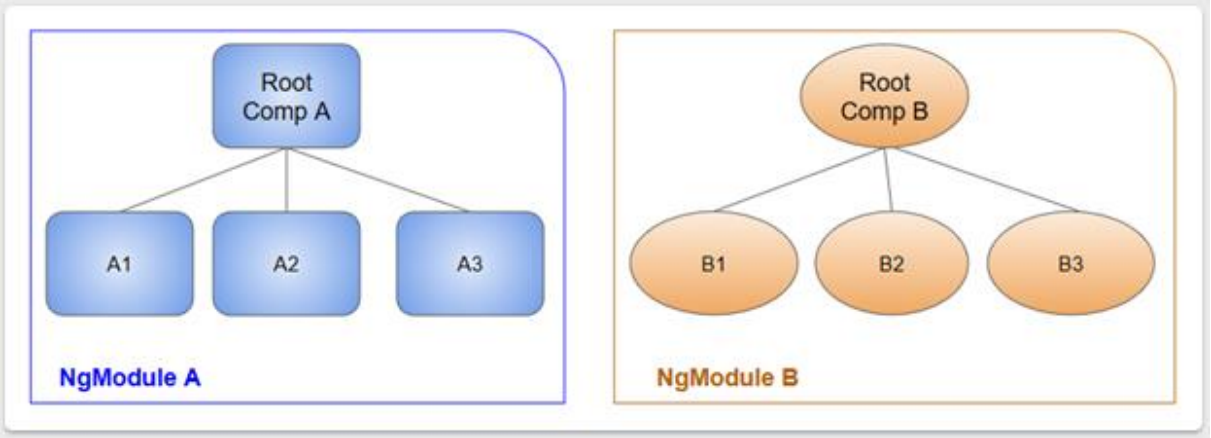

Figura 6 Angular Modules (Google, s.d.)

4.1.3 Componenti

I componenti son i blocchi base della programmazione di applicativi Angular dato che modella un elemento grafico o un comportamento e può contenere a sua volta altri componenti. Questi sono costituiti da tre file principalmente:

1. HTML: un template che definisce gli elementi che saranno presenti nella pagina.
2. TypeScript: un modello logico che definisce il comportamento degli elementi definiti nel html.
3. CSS: una descrizione locale degli elementi che sono stati definiti nel template. Questa descrizione può rappresentare anche un'estensione o una specificazione del CSS ereditato.

Questi tre file sono legati grazie al decorator @Component() presente nella classe definita nel file TypeScript, che riceve come input un oggetto con metadati, fra questi "*selector*", "*templateUrl*", "*styleUrls*" e altri. *TemplateUrl* e *styleUrls* corrispondono rispettivamente ai path dove il file html e css corrispondenti al componente sono presenti.

4.1.4 Servizi

I servizi Angular sono classi che contengono logiche e funzionalità specifiche per la gestione e il flusso dei dati. In pratica i servizi contengono insiemi di funzioni che si possono usare in qualunque modulo evitando la replicazione del codice e permettendo di accentrare la gestione in un solo punto. In questo modo, i moduli possono delegare ai servizi certi lavori che non fanno parte della funzionalità specifica di ogni modulo e che normalmente sono uguali per tutti come l'acquisizione di dati dal server o la validazione degli input. Questo processo è permesso grazie al concetto di servizi "*injectable*": non è altro che concedere ai componenti di usare funzionalità contenute al suo esterno solo nel caso siano state disegnate a questo proposito. Queste classi usano il decorator @Injectable() e, per essere usate all'interno dei componenti, devono essere dichiarate come input nel costruttore del componente.

4.2 Descrizione della soluzione

La scelta grafica dei componenti originali è stata presa da una delle palette di colori messa a disposizione dall'utente per la personalizzazione delle pagine interne. Per motivi di privacy, i colori sono stati ridefiniti e non rappresentano nessun vincolo con nessuna azienda. Inoltre, per questo progetto si sono adeguati alcuni dei componenti che si sovrapponevano al perimetro di funzionalità di altri applicativi, quindi alcuni requisiti preesistenti sono stati cancellati o ridefiniti per adeguarsi al funzionamento base voluto nell'applicativo. Questa decisione è stata presa perché il cliente ha comunicato di non avere nessuna intenzione di modificare il perimetro degli applicativi esistenti per questione di organizzazione e per evitare i problemi di ridondanza dei dati.

4.2.1 Moduli

QDM è un applicativo semplice che ha bisogno di accentrare l'informazione necessaria per rendere il processo di controllo di qualità molto più semplice e veloce. Quindi, per sviluppare il layer di Front-End si è pensato di creare un modulo principale unico con un componente che rappresenti un cruscotto o dashboard. Questo componente conterrà all'interno i diversi widget divisi per argomento. Ogni uno dei widget verrà creato in un componente a parte che userà altri componenti disegnati per la rappresentazione grafica dei dati e che sono generici per il riutilizzo.

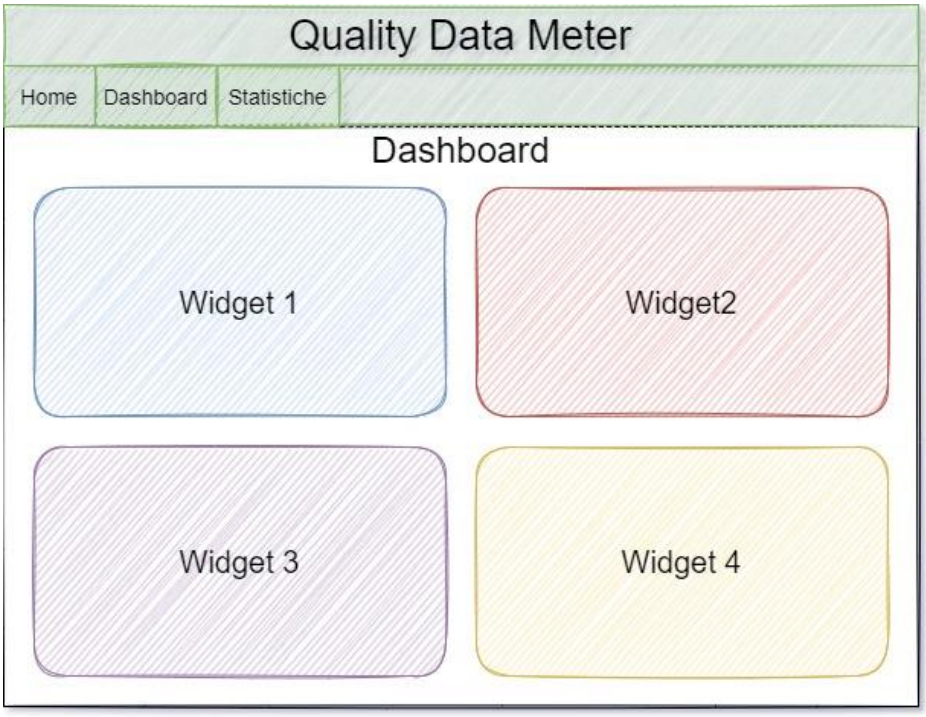

Figura 7: Bozza Dashboard QDM

Si prevede anche la creazione di un modulo di Routing per permettere la navigazione fra gli elementi disposti nella barra orizzontale di navigazioni per i quali si prevedono componenti diversi ma allo stesso livello della dashboard. Questo modulo di Routing permette, anche avendo una Single Page Application, di navigare in pagine con URL diverse all'interno dello stesso applicativo. In questo modo non è necessario caricare tutto il contenuto dei diversi tab ma solo quelli del tab corrente e si preservano le performance dell'applicativo Angular.

Inoltre, si è pensato di incapsulare in un modulo unico tutte le funzionalità trasversali alle quali si vuole fare accesso in modo pubblico, statico e generico come lo sono le trasformazioni di dati o la loro formattazione.

4.2.2 Componenti

Nel progetto si trova una grande quantità di componenti che modellano e dettagliano tutti i componenti dell'applicativo, perciò si è disegnato un sistema gerarchico e tematico per

organizzarli. In cima si trova il componente applicativo padre che contiene l'informazione del header e il menu orizzontale con il rispettivo routing. I componenti che compongono questo menu sono:

- Home: pagina di benvenuto molto semplice che non ha nessuna rilevanza per il progetto.
- Dashboard: pagina dove si trovano i widget con il rispettivo servizio "Mocked" che fornisce dati preimpostati per valorizzare il widget.
- Statistiche: pagina che permette la visualizzazione dei dati "mocked" in modo di grafici e diagrammi.
- Portale: pagina simile alla Dashboard che permette la visualizzazione dei widget ottenuta dal layer di Back-End, quindi con dati verosimili e presenti nella base di dati.

Queste voci del menu orizzontale sono bottoni che indirizzano la porzione principale della pagina verso il componente rispettivo (verranno descritti ed illustrati nella sezione ***3.3 Componenti principali***). Ognuno di questi componenti, a sua volta, è composto da widget. I widget sono componenti che rappresentano un insieme di informazioni utili all'utente, che rispettano l'organizzazione tematica decisa in questo progetto e che sono integrati da ulteriori componenti che rappresentano dati specifici.

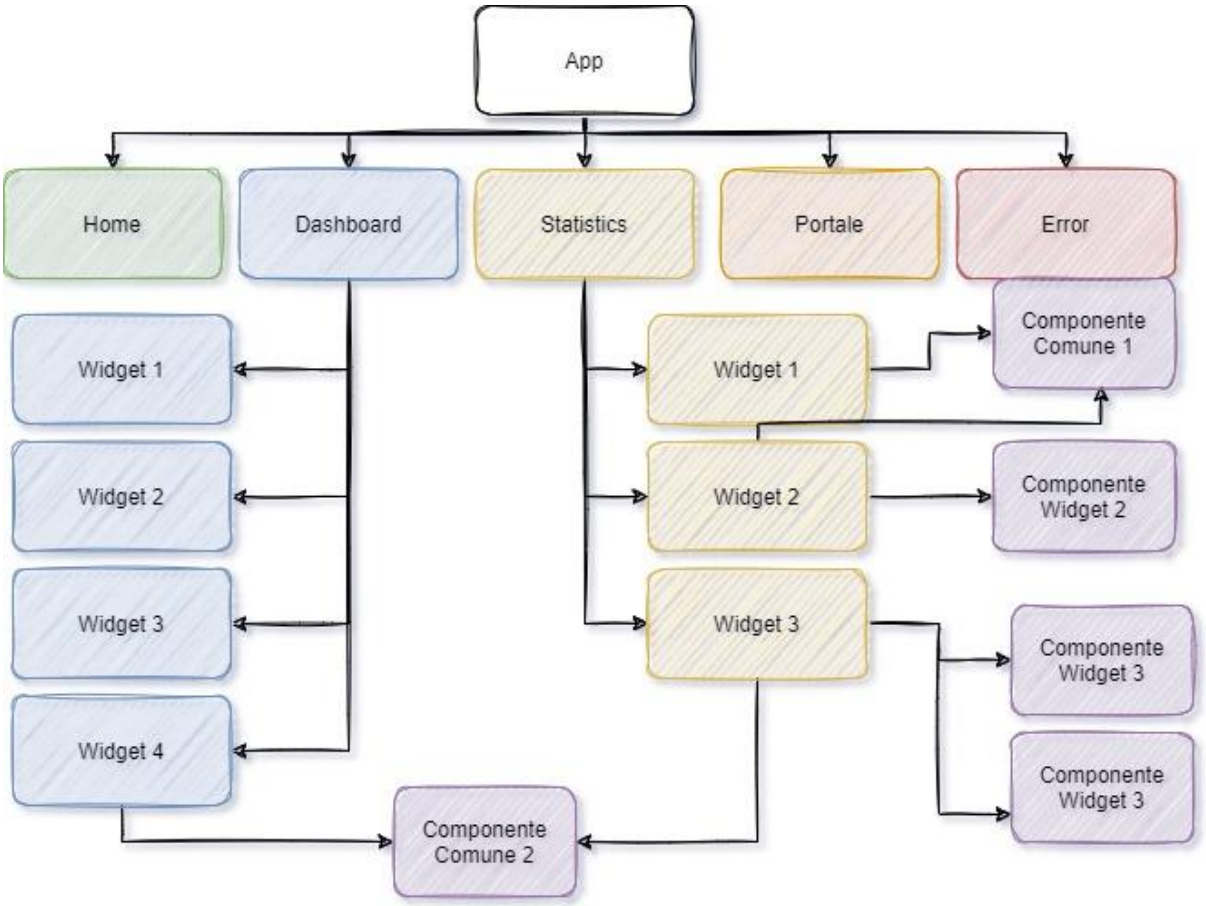

Figura 8: Diagramma componenti

Per standardizzare la creazione dei widget, è stato creato un componente chiamato top-widget che definisce il comportamento e l'aspetto generale di essi. In questo modo, si rende uniforme l'esposizione di questi elementi e si facilita la loro manutenzione e le future nuove creazioni.

Per concretizzare le idee esposte precedentemente di seguito saranno presentati i concetti con relazione alle immagini che si possono trovare nell'applicativo:

- Dashboard: è uno dei componenti di secondo livello che contiene alcuni widget.

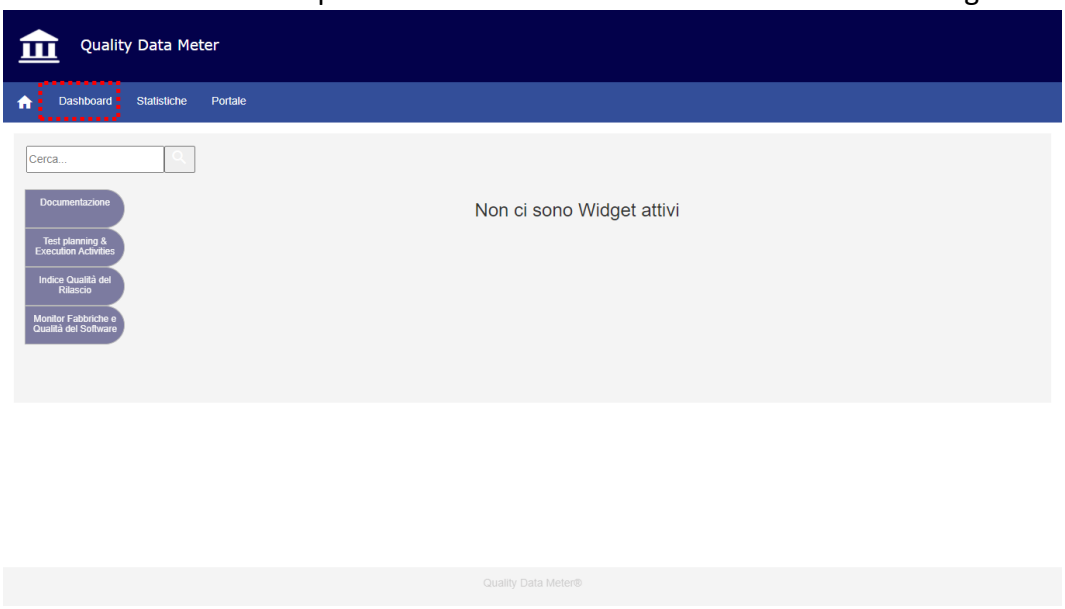

Figura 9: Componente secondo livello

- Widget: I widget sono quei segmenti della pagina che si selezionano per visualizzare i dati. In questo caso si hanno due widget aperti.

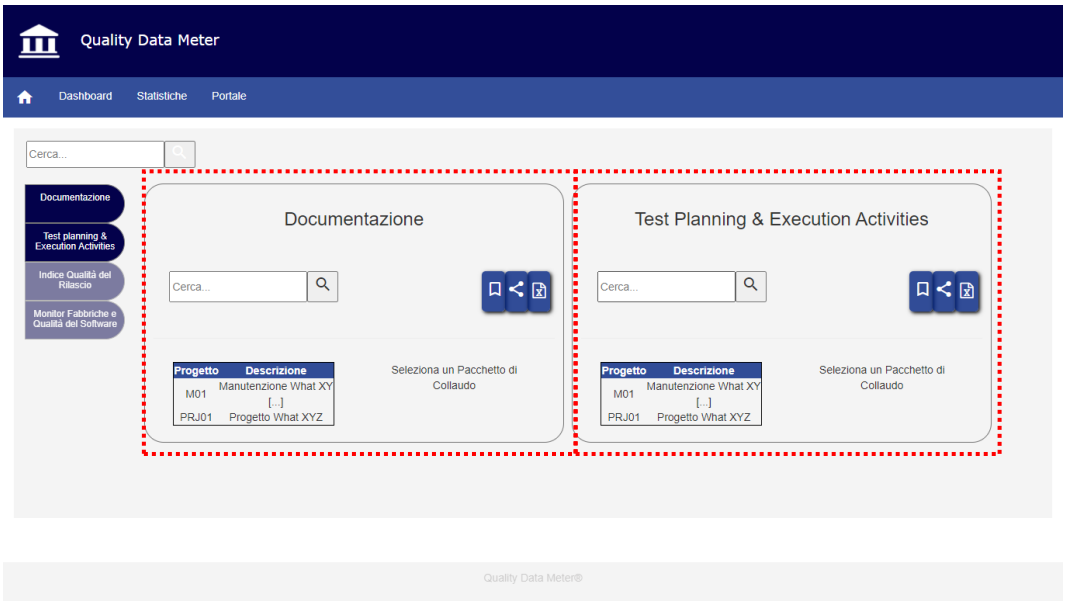

Figura 10: Componente Widget

- Componenti comuni: I widget contengono componenti comuni che sono definiti al di fuori della loro posizione. In questo caso troviamo una tabella che elenca i progetti e che è presente in entrambi i widget

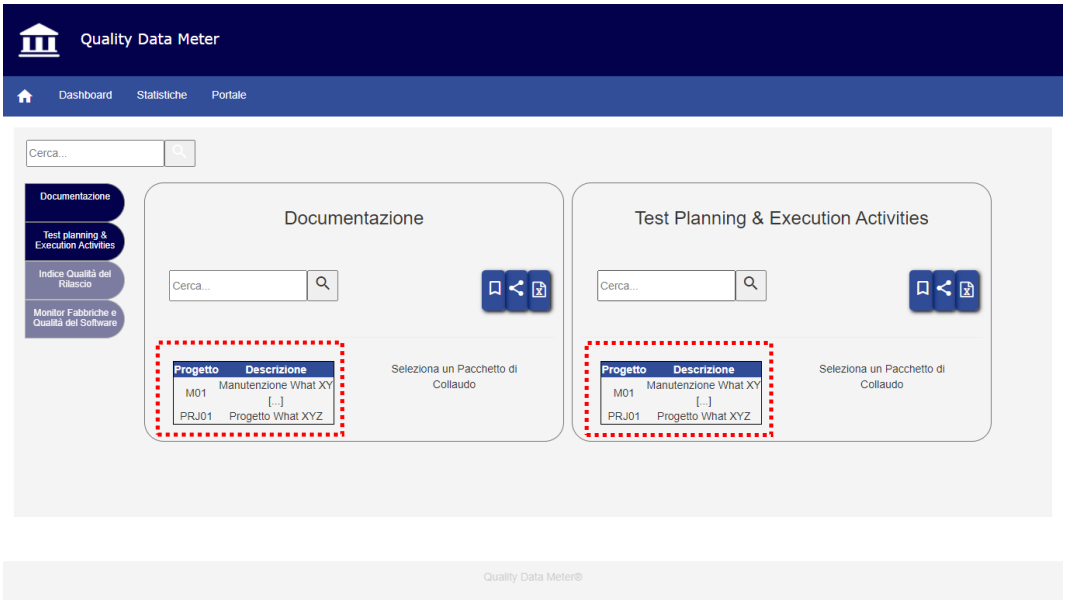

Figura 11: Componente comune

- Componente specifico: i widget contengono anche componenti specifici che sono utili soltanto al loro interno, quindi, vengono definiti sotto la loro alberatura. In questo caso abbiamo un modello di punti che indicano la presenza o meno di un documento all'interno di uno step specifico.

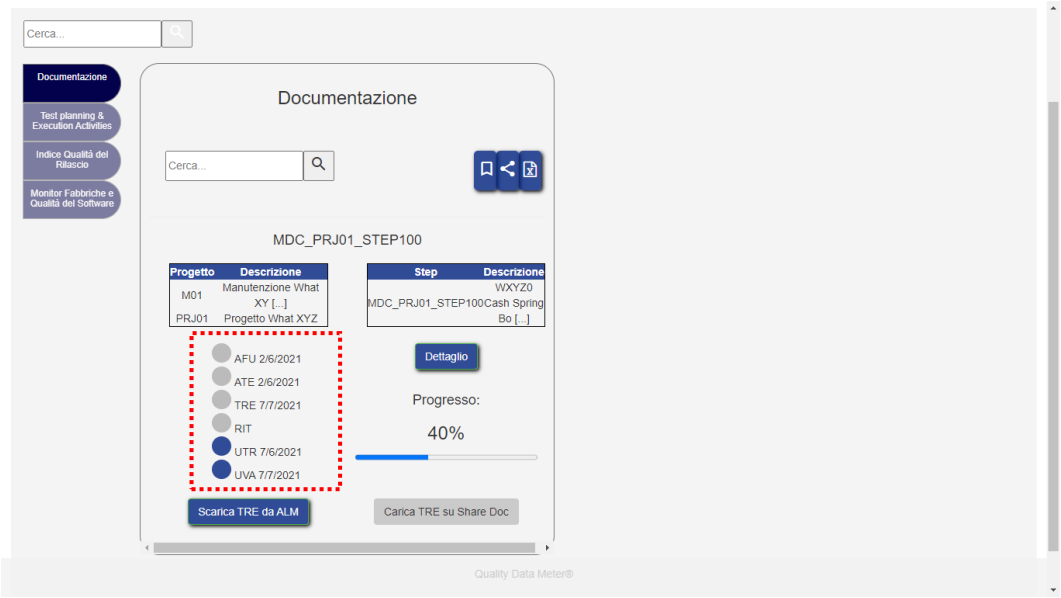

Figura 12: Componente specifico

- Sottocomponenti: ogni componente può contenere altri componenti che lo aiutino ad essere meno verbosi. Quindi, in questo caso, si è creato un componente ancora più specifico che contiene un cerchio che viene colorato o grigio, una descrizione di documento e una data relativa al documento.

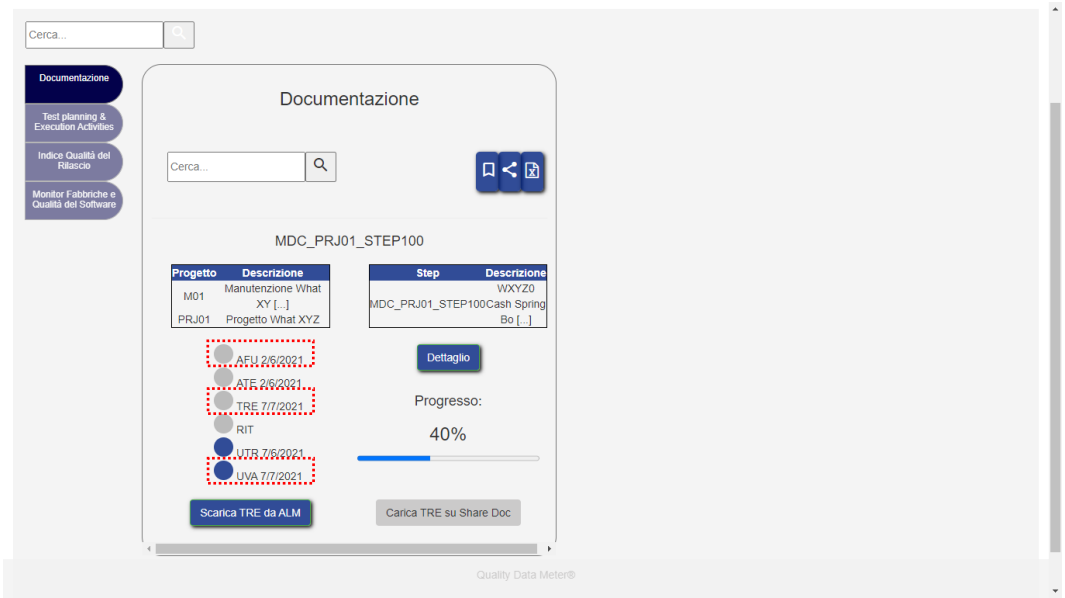

Figura 13: Sottocomponente

4.2.3 Servizi

Per favorire la semplicità interna dei componenti e per accentrare la gestione dei dati e la loro acquisizione si sono creati due servizi diversi: Data Service e Call Service.

Il Call Service corrisponde al servizio che si interfaccia con il layer di Back-End e ha come obiettivo costruire le richieste REST con formato Json da fare al server e fornire la risposta data al secondo servizio. Nel caso di cambio di parametri per contattare il server, le modifiche si limitano a questo servizio dato che è l'unico che le conosce ed è l'unico punto in cui si interfaccia con l'altro layer.

D'altra parte, si trova il Data Service che corrisponde ad un servizio contenente una serie di variabili che accentrano l'informazione generica utile ai widget, quindi liste di oggetti che modellano i dati da mostrare a video. In più, questo servizio contiene oggetti di tipo "*EventEmitter*" che permettono il flusso di informazione con il pattern Publish/Subscribe dal Call Service e verso i componenti. Questo servizio prende la risposta fornita dal Back-End e la mappa nei diversi contenitori dati per poi inviarli ai componenti. Si è deciso di creare questo servizio per evitare chiamate inutili al server e per condividere informazioni comuni fra i componenti.

È importante capire che i modelli dati sono stati accentrati in classi (Data Transfer Objects) dentro una cartella models che è accessibile a tutte le classi del progetto quindi i servizi e i componenti possono scambiare informazioni liberamente usando i modelli adeguati.

4.3 Componenti principali

Conoscendo i requisiti e sapendo come sono stati disegnati i componenti Angular per l'implementazione del layer di Front-End, a continuazione si descriverà il comportamento e la proposta grafica per il progetto di Quality Data Meter.

4.3.1 Dashboard

La dashboard è la pagina che permette la visualizzazione dei quattro widget. L'informazione fornita a questi widget è fittizia (mocked) e si trova in diversi file Json all'interno del progetto del Front-End. Questa pagina è stata disegnata fin dall'inizio con l'obiettivo di mostrare un prototipo funzionale all'utente prima dell'elaborazione del front-end. La creazione dei componenti interni e le logiche dei widget sono state elaborate all'interno di questo componente e dopo sono state riutilizzate nella sezione Portale.

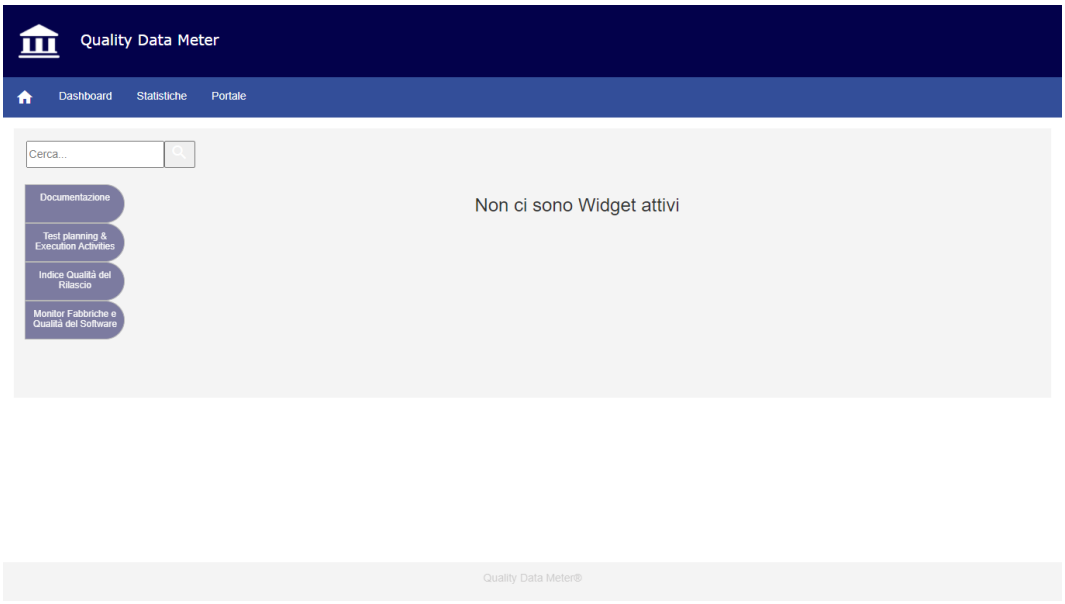

Figura 14: Dashboard-Schermata Iniziale

Inizialmente la dashboard si mostra vuota (con tutti i widget chiusi) e contiene un menù laterale che permette l'avvio dei widget.

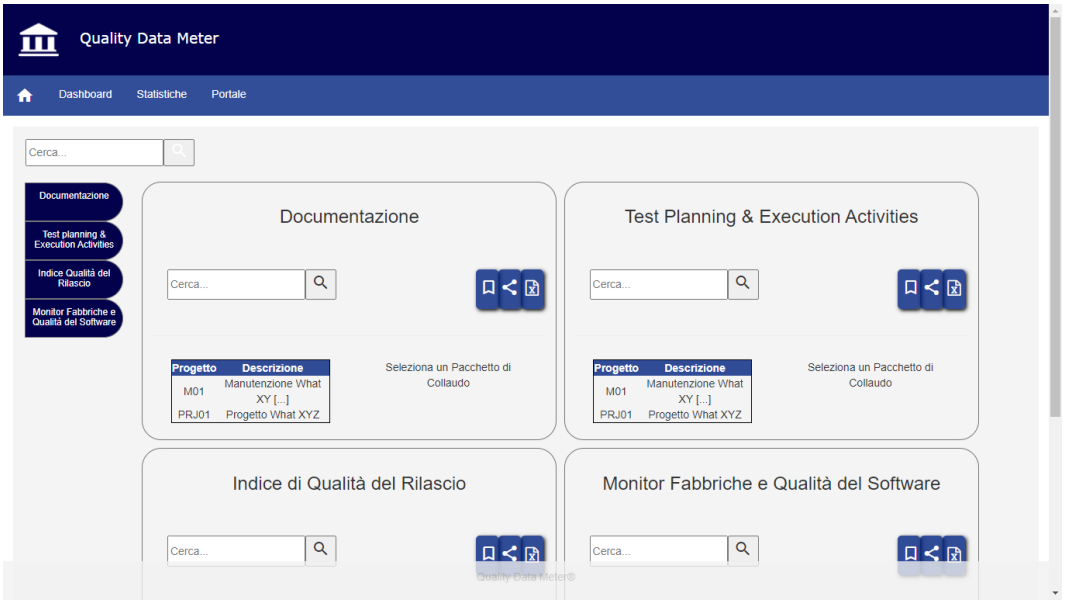

Figura 15: Dashboard-Impostazione Widget

I widget sono organizzati ciascuno in un quadrante di uguale grandezza e, con il menu laterale, si possono chiudere e riorganizzare.

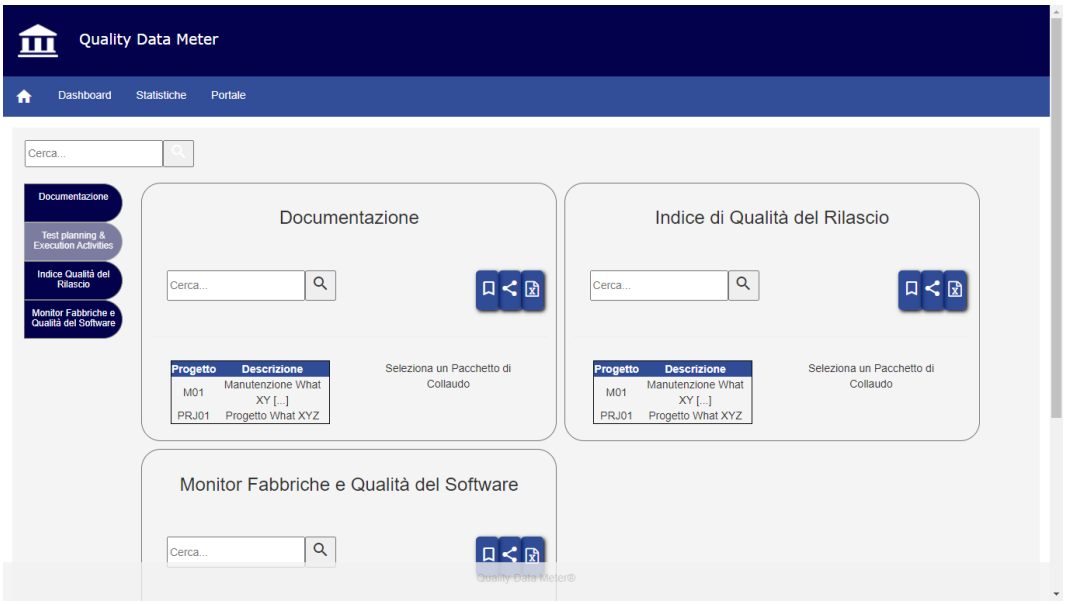

Figura 16:Dashboard-Widget riorganizzati

Questa pagina è stata sviluppata in modo da permettere la visione contemporanea dei quattro widget e si adatta in modo verticale quando lo schermo raggiunge una larghezza minima.

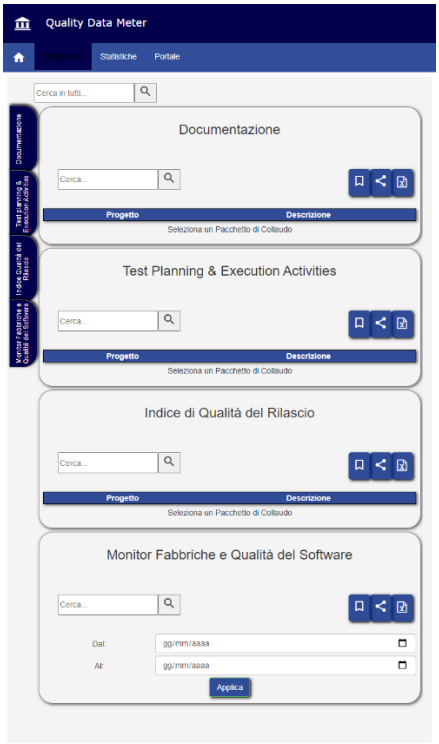

Figura 17:Dashboard-Riorganizzazione verticale

4.3.2 Statistiche

La pagina di Statistiche è stata disegnata a scopo puramente illustrativo per mostrare al cliente che questo progetto non sarebbe soltanto un punto di raccolta di dati ma potrebbe essere espanso anche per l'analisi di molti altri fattori e indici con grafici dinamici. A livello tecnologico, sono stati usati chart della libreria Bootstrap versione 3.3.7. La pagina mostra direttamente tre widget grafici tutti rivolti ad uno specifico ufficio: i primi due hanno informazione statica che rappresenta la percentuale di completamento della documentazione dei diversi progetti associati e l'andamento dei costi e la qualità.

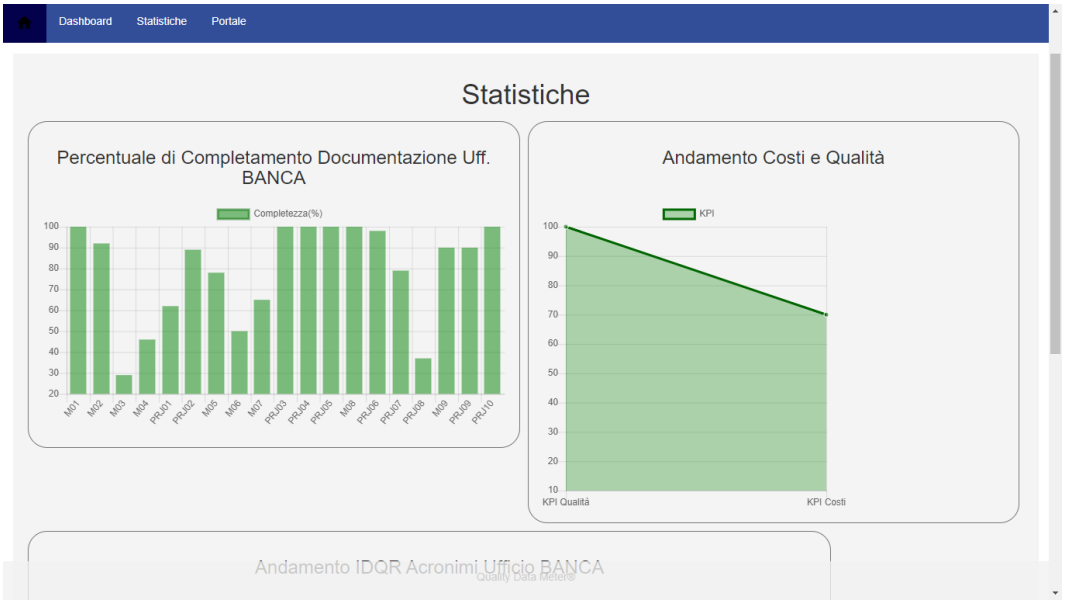

Figura 18: Statistiche-Widget statici

D'altra parte, il terzo widget mostra in modo dinamico l'andamento temporale dell'indice di qualità medio per ogni acronimo associato all'ufficio con un grafico e una tabella.

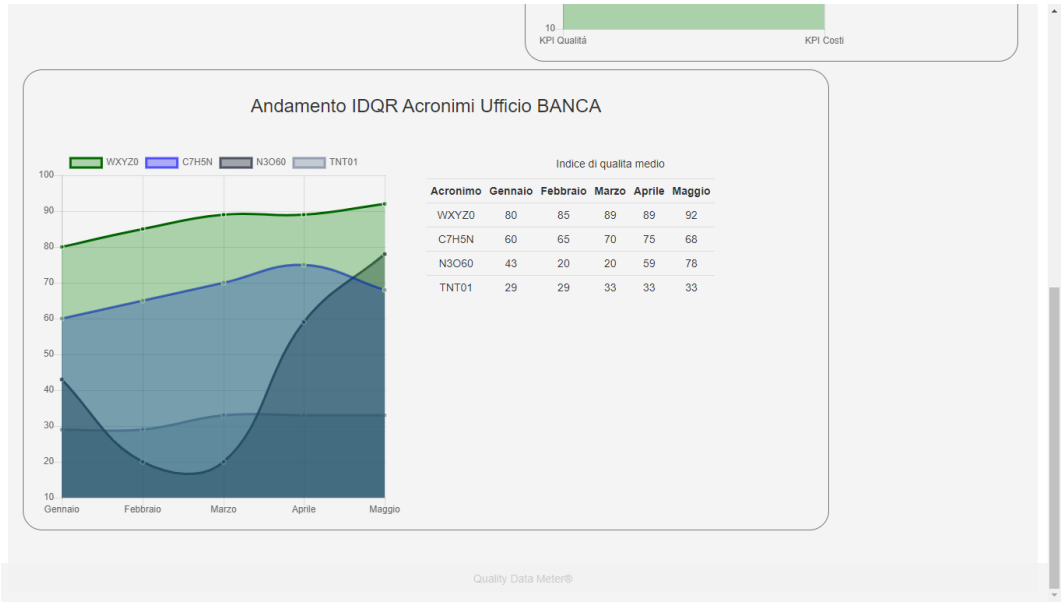

Figura 19:Statistiche-Widget dinamico

Questo ultimo widget permette un'interazione fra i due componenti principali: facendo click in una riga della tabella che rappresenta un acronimo, il grafico si aggiorna per evidenziare l'andamento del segmento voluto e dare un dettaglio più specifico, come si vede nella seguente immagine.

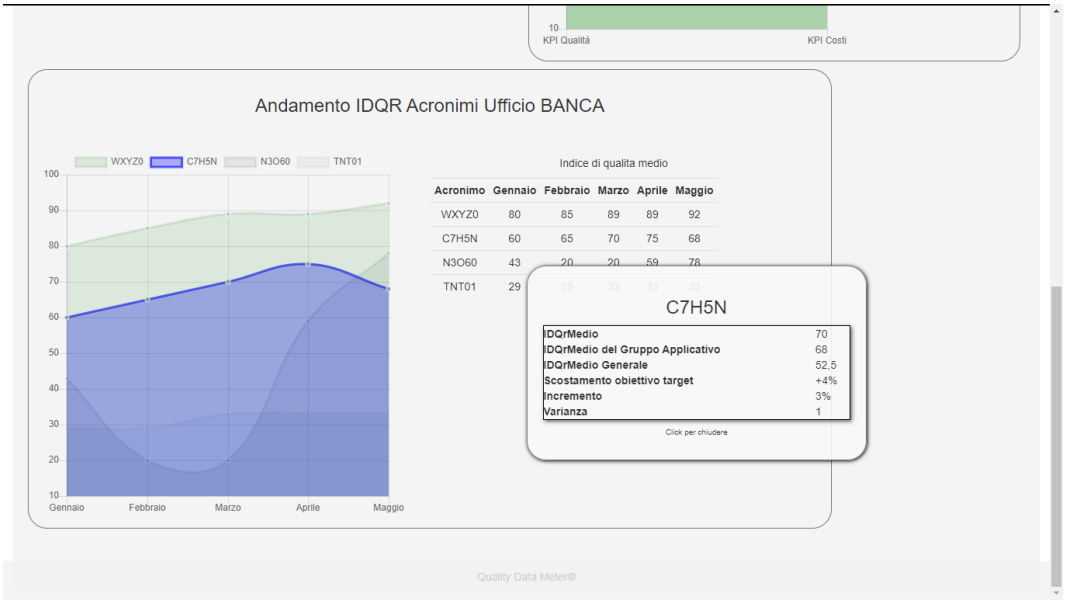

Figura 20: Statistiche-Widget Dinamico Filtrato

4.3.3 Portale

La pagina di portale è stata creata riutilizzando le logiche e i componenti fatti per la Dashboard ma sono stati adeguati a contenere informazioni reali provenienti dal layer di Back-End che a sua volta le ricava dalla base di dati.

L'impostazione iniziale dei widget è opposta a quella della Dashboard perché agli utenti farebbe più comodo avere sempre la visione di tutti i widget e, in casi particolari, toglierne alcuni per ridurre l'informazione visualizzata. Inoltre, se il backend non è pronto oppure non ci sono dati da mostrare, il portale mostra i widget vuoti senza la possibilità di fare nessun click.

Il menu laterale è stato messo in senso verticale per massimizzare lo spazio orizzontale per i widget e permettere di mostrare le tabelle in modo ottimale.

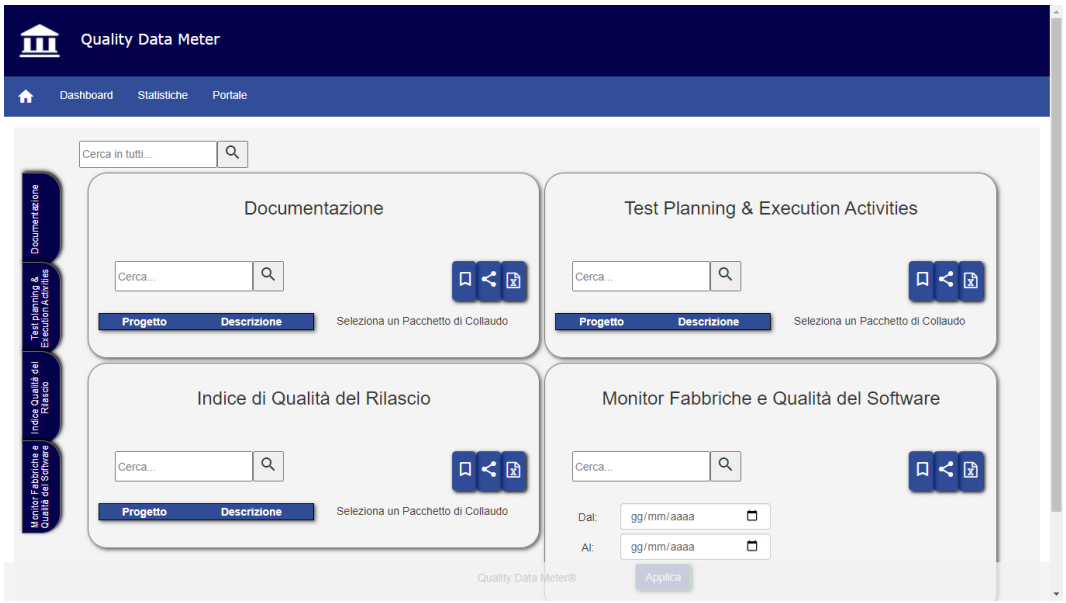

Figura 21: Portale- Schermata Iniziale

4.3.4 Widget Documentazione

Il widget di Documentazione si basa sul requisito che richiede al sistema di permettere il controllo dei dati dei documenti forniti nelle diverse fasi del processo di creazione e modifica dei progetti software del cliente. Questo widget deve offrire un'interfaccia grafica che permetta agli utenti di capire la percentuale di completamento documentale tenendo in considerazione solo i documenti obbligatori per ogni progetto, le date di caricamento di ogni documento e i dettagli del progetto. La visualizzazione di questi elementi deve essere raggruppata per Progetto e Step in modo da avere una visione locale.

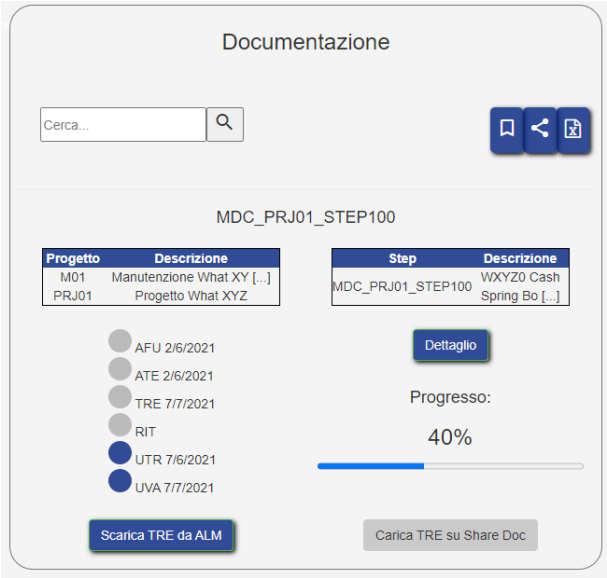

Figura 22: Widget Documentazione

4.3.5 Widget Test Planning & Execution Activities

Il widget di Test Planning permette la visione della pianificazione dei test e i dati dell'esecuzione degli stessi per ogni step in ogni progetto. Inizialmente si era previso un vincolo con il sistema attuale di gestione dei test in cui dal progetto QDM era possibile caricare dei dati di testing per il calcolo delle percentuale e il tracciamento delle attività. Questa funzionalità è stata eliminata completamente e si è introdotto soltanto un bottone di reindirizzamento al sistema di gestione dei test attuale. Inoltre, internamente si è aggiunta una funzione per calcolare la percentuale di completamento che dipenda soltanto dai test eseguiti e passati.

Attualmente si prevede il tracciamento soltanto dei test manuali ma, nel framework e nei protocolli che si stanno cominciando ad usare, si prevede anche la creazione ed esecuzioni di test automatici e semi-automatici i cui output potrebbero rappresentare un'altra fonte di dati per questo progetto.

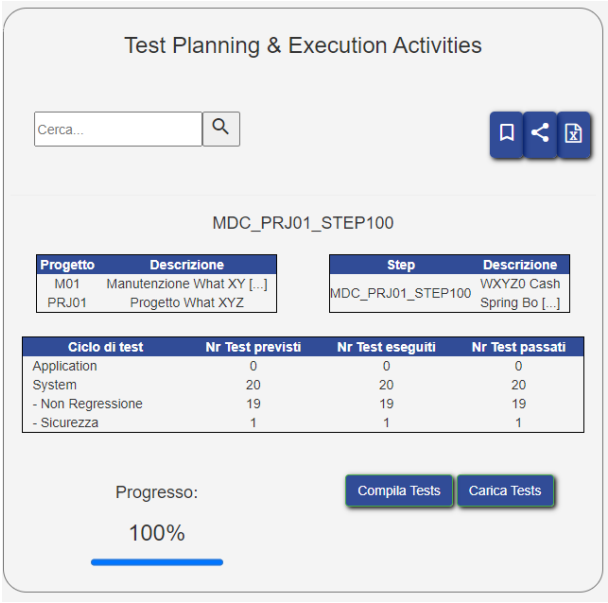

Figura 23: Widget Test Planning & Execution Activities

4.3.6 Widget Indice di Qualità del Rilascio

Il widget legato all'indice di qualità del rilascio permette la visualizzazione dell'indice principale calcolato da tutti i parametri dei diversi argomenti. Inoltre, mostra l'indice di qualità medio per lo step legato all'acronimo, la quantità di giorni mancanti per il rilascio e i dati di dettaglio principali come i responsabili del processo e le date relative ai passaggi nei diversi ambienti.

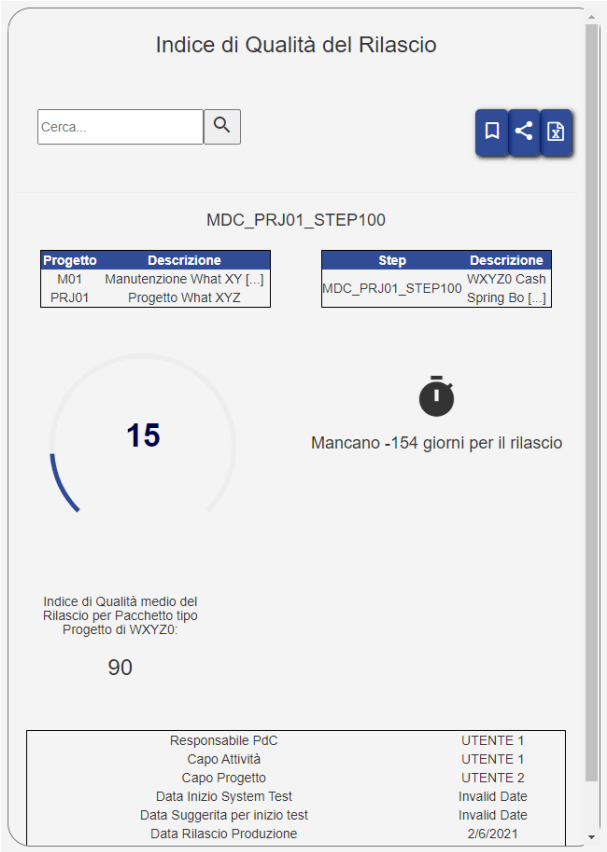

Figura 24: Widget Indice di Qualità del Rilascio

In aggiunta, originalmente si era previsto un bottone per il blocco/sblocco del rilascio che, interfacciandosi con il sistema di Change, permettesse agli amministratori di fermare tutto il processo motivandone la ragione ma, con l'adeguamento e le specifiche ulteriormente dettagliate, si è deciso di inabilitarlo per due motivi: funzionalità fuori perimetro (interferenza con i meccanismi forniti dalla Change Console) e non profilazione degli utenti (si è previsto un solo tipo di utente per azioni di sola visualizzazione).

4.3.7 Widget Monitor Fabbriche e Qualità del software

Il widget Monitor Fabbriche e qualità del software permette la visione a grande scala delle attività e le date più importanti in un intervallo dato. La schermata iniziale del widget mostra un filtro di date che deve essere selezionato per mostrare un calendario con i giorni importanti per i rilasci. Sotto si mostra un dettaglio numerico delle diverse attività visibili nel calendario, un indice di qualità medio del rilascio e diversi bottoni che presentano dettagli di costi, ticket e qualità. Questi dettagli si mostrano in finestre modali in modo tabellare e rappresentano un riepilogo dei diversi rilasci e dei progetti ad essi associati.

Figura 25: Widget Monitor Fabbriche e Qualità del software

Capitolo 5 Implementazione del Back-End

La fase più estesa e complessa è stata la definizione del Back-End. Avendo come modello architetturale i microservizi, è sato necessario definire quanti e quali microservizi si devono sviluppare. Inizialmente si è pensato che il progetto si potesse definire in due moduli: uno che fosse dedicato all'acquisizione e il processamento dei dati (Engine) ed un altro che fornisse un'interfaccia verso il client per l'accesso e la gestione dei dati (Services). In una fase di analisi posteriore si è deciso di frammentare ulteriormente il modulo dei Services dato che, per i requisiti di sistema, si deve puntare ad una soluzione flessibile, scalabile e facilmente mantenibile. A questo punto era necessario capire quale era il miglior modo di dividere i servizi. La prima opzione era quella di creare un microservizio per ogni widget esposto nel portale, la seconda era di suddividere i servizi per tematica (dominio di dati). La soluzione doveva essere quella più snella e quella che potrebbe dare maggiori vantaggi al sistema. La prima soluzione darebbe un isolamento definito dall'interfaccia grafica, invece la seconda lo darebbe a seconda del dominio dei dati che si vogliono acquisire. Dunque, la prima soluzione è stata scartata dato che, come si può vedere nel capitolo precedente, i widget hanno molti dati in comune (tutto il perimetro dei progetti e step) che permettono di dare un ulteriore dettaglio a seconda dell'argomento. Questo stesso pensiero ha portato l'idea che suddividere i servizi per argomento (dominio di dati) era la migliore soluzione dato che è desiderabile evitare la duplicazione del codice ed è ideale che, a futuro, se dovessero venire introdotti nuovi dati nel modello di business, i servizi da introdurre potrebbero essere creati da zero senza nessuna dipendenza da quelli già esistenti.

Il linguaggio di programmazione scelto dall'azienda per l'implementazione dei servizi è Java, un linguaggio di programmazione ad alto livello, orientato agli oggetti e uno dei più diffusi al mondo.

5.1 Definizione tecnologiche

Per il progetto di QDM si sono definiti certi framework e tecnologie che permetteranno di sviluppare in modo uniforme i diversi microservizi da creare. Questi strumenti, correttamente organizzati ed impostati, aiutano all'automatizzazione dei processi e a minimizzare le implementazioni degli sviluppatori, dando priorità alle funzionalità specifiche del progetto.

5.1.1 Spring framework

Si è deciso di usare il framework Spring, molto diffuso a livello mondiale per la creazioni di applicazioni Java. In particolare, si è scelto di usare Spring boot per facilitare la creazione dei microservizi con l'aiuto di tutte le funzionalità e i concetti che sono implementati nel framework. In più, della stessa famiglia, è stata adottato anche il framework Spring Data che permette la semplificazione nella creazione di entità e repositories grazie a dei descrittori che attivano le funzionalità delle librerie. Per ultimo, in particolare per il modulo Engine, si è deciso

di usare la libreria Quartz appoggiata al framework Spring che permette lo scheduling di job in modo semplice e ordinato.

5.1.2 Maven

Per gestire i progetti e le loro dipendenze, è stato usato il tool Apache Maven. Basato sul concetto di Project Object Model (POM) Maven permette di descrivere il progetto in modo statico, le sue dipendenze verso librerie, plugins utili per i processi di build, testing, deploy e molto altro. In particolare, questo strumento è stato di particolare utilità grazie all'integrazione di un plugin Docker che permette di automatizzare il processo di deploy locale.

5.1.3 Docker

Piattaforma di esecuzione per l'automatizzazione della distribuzione di applicativi software all'interno di contenitori, appoggiandosi nel concetto di virtualizzazione. Questa piattaforma ha reso possibile creare diversi contenitori con i moduli del progetto per permettere di fare un deploy completo di tutto il sistema, compresa la base di dati (presente anche come container Docker e resa disponibile ai diversi microservizi grazie ad una rete virtuale locale). Le immagini di JBoss e Oracle DB usate sono le immagini ufficiali che si trovano nel repository Docker-Hub.

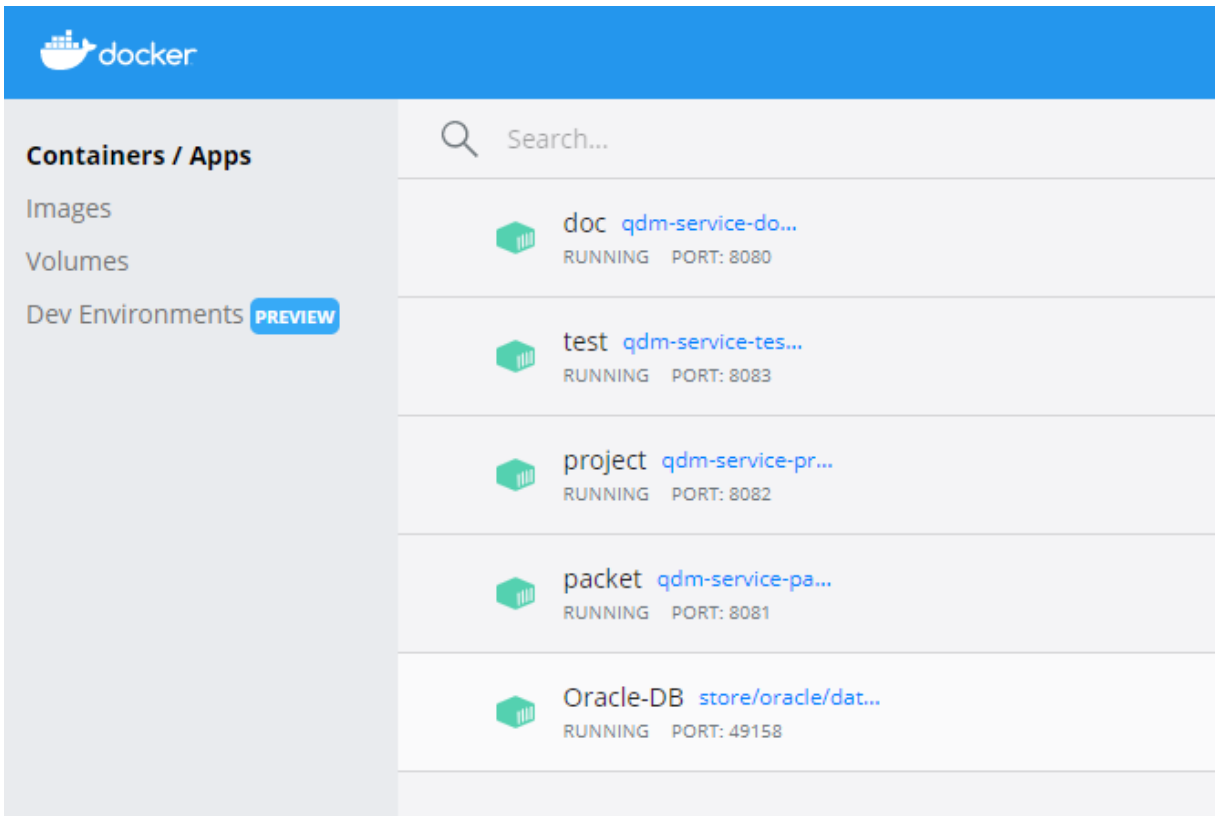

Figura 26: Container Docker

Nell'immagine si possono vedere i container dei diversi moduli progettuali corrispondenti ad ogni microservizio. Il nome principale è un alias che permette l'identificazione del container in modo veloce, il nome del microservizio si trova nella parte desta in colore azzurro con la porta corrispondente sotto. Inoltre, si può vedere il container del Oracle DB che viene usato dai microservizi ed è l'unico proiettato per questa soluzione. A futuro si potrebbe pensare a nuove base di dati presenti in altri container e una vasta quantità di microservizi con funzionalità diverse.

Grazie al plugin Maven fabric8io, si può personalizzare ed automatizzare il processo di deploy con Docker for Windows. Infatti, tutti i progetti contengono una configurazione standard che facilita il processo e rendono lo sviluppo e il testing applicativo molto semplici.

5.2 Definizione dei moduli progettuali

5.2.1 Libreria Core

In aggiunta ai moduli proiettati nella soluzione, si è pensato di creare una libreria di nome qdm-core che abbia in modo accentrato le funzionalità più generiche, i modelli di entità che permettono l'accesso alla base di dati, alcuni repository che contengano le query più generali per l'accesso ai dati e classi di utilità trasversali. Questo modulo viene solo istallato e usato come dipendenza Maven all'interno di ogni microservizio. Se questo modulo non fosse sufficiente per qualche microservizio, le funzionalità si potrebbero estendere e sarebbe possibile implementare ulteriori repository all'interno di ogni progetto per coprire qualsiasi necessità. Il modulo core deve evolvere con il sistema in modo di non interferire con la funzionalità dei vecchi microservizi ma garantendo l'integrità dei dati.

5.2.2 Definizione servizi Engine

Il modulo qdm-engine è un progetto disegnato per verificare periodicamente l'esistenza di flussi di dati di input che devono essere acquisiti, processati e salvati. Specificamente questo progetto cerca dei file XML contenenti i dati forniti dai sistemi alimentanti per essere letti, elaborati e scritti sulle tabelle nella base di dati Oracle per renderli disponibili. Questo progetto ha in carico il flusso dell'informazione da quando dei processi forniti dall'azienda ospitante scodano i messaggi scrivendogli nel filesystem del contenitore fino al rilascio dei dati nella base di dati del progetto e la posteriore cancellazione del file XML.

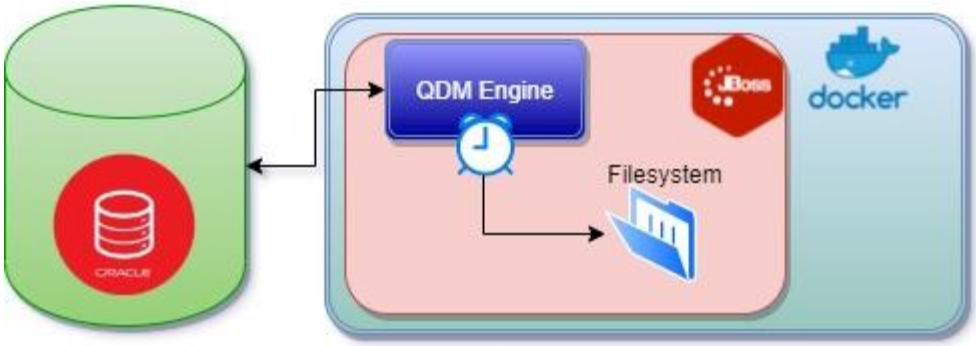

Figura 27: Modulo engine

Ad ogni file XML previsto come input per il sistema corrisponde una classe Parser, che traduce l'informazione dal formato descrittivo all'entità relazionale disegnata specificamente per rappresentare questo oggetto. Sono stati previste otto tipologie di file XML che sono distinti per il nome del file (a seconda di un prefisso, l'XML corrisponde ad una serie di dati diversi). Questa distinzione è stata decisa analizzando un sottoinsieme dei dati visibili nei flussi dei sistemi alimentanti ma sono facilmente modificabili e i Parser associati sono, anche essi, modificabili ed implementabili facilmente.

Questo servizio è replicabile facilmente dato che gestisce dei lock nei record della base di dati e nei file XML che gestiscono la concorrenza e in più gli scodatori disegnati dall'azienda permettono la distribuzione del carico in modo dinamico a seconda di quanti container sono disponibili. Essendo un progetto sviluppato per funzionare con dei container JBoss nella piattaforma Docker, la messa in azione di un container è semplicemente il commando "Run" di un'immagine del Application Server con dei parametri specifici per permettere l'interconnessione con la base di dati e la disposizione di un servizio ftp disponibile per gli scodatori. In questo modo, lo scale up per il processamento dei dati viene semplificato. Si deve tenere in conto una limitazione data dall'implementazione degli scodatori: i tipi di messaggi che vengono scodati possono essere misti, quindi il microservizio di traduzione deve contenere tutti i parser corrispondenti a tutti i messaggi possibilmente scodati.

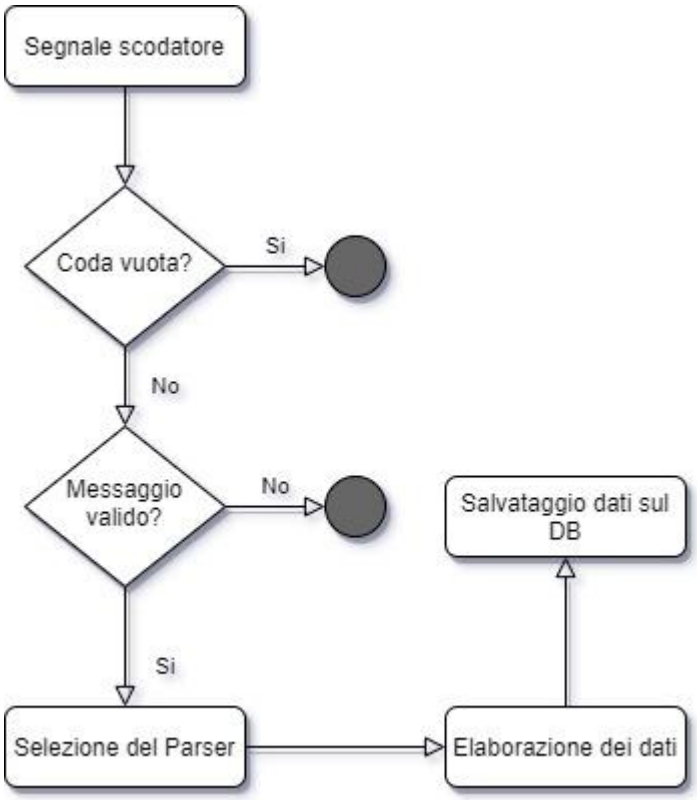

Figura 28: Diagramma di flusso Engine

5.2.3 Definizione microservizi

La definizione dei microservizi, come si è detto precedentemente, è stata fatta a seconda del perimetro dei dati che deve gestire. Si sono individuati quattro concetti principali per l'implementazione che si vuole sviluppare (sicuramente in un futuro dovrà essere estesa per coprire tutte le caratteristiche che l'area di lavoro ingloba) e quindi quattro microservizi da implementare:

1. qdm-service-project: corrisponde all'insieme di servizi che forniscono l'informazione legata ai progetti in modo generico.
2. qdm-service-packet: corrisponde all'insieme di servizi che forniscono l'informazione legata ai pacchetti e gli step che lo compongono.
3. qdm-service-test: corrisponde all'insieme di servizi che forniscono l'informazione riguardanti i dati di testing.
4. qdm-service-documentazione: corrisponde all'insieme di servizi che forniscono l'informazione riguardante la documentazione e tutte le date che sono associate alla consegna dei documenti.

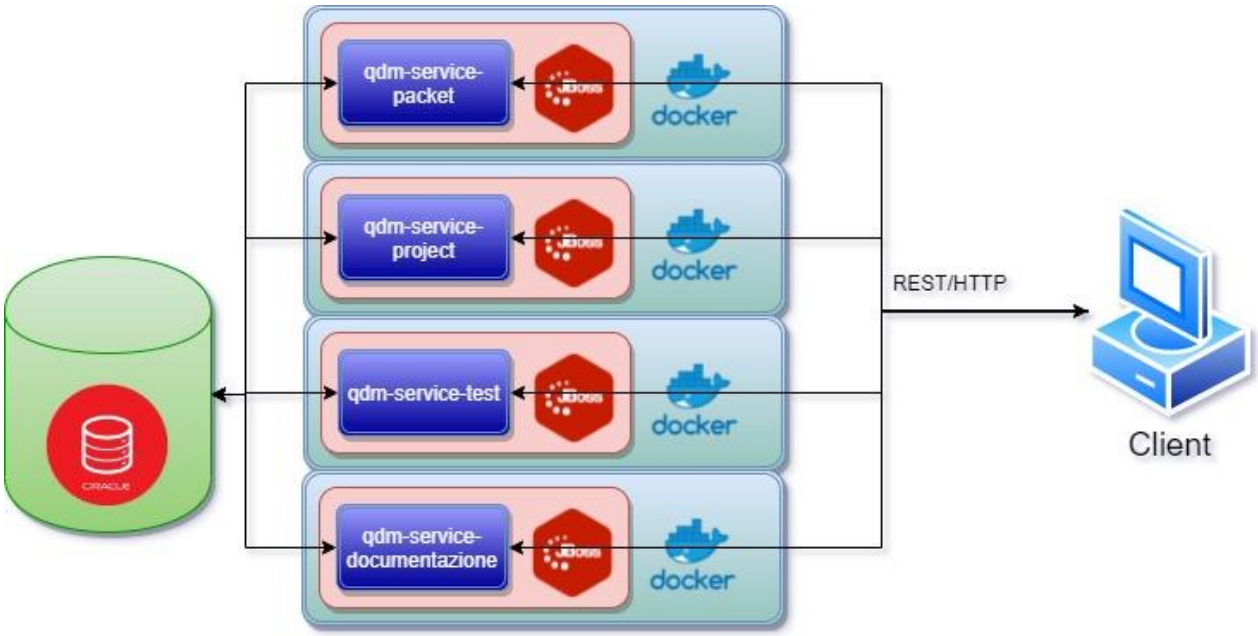

Figura 29: Moduli services

5.2.4 Definizione della Base di Dati

Il sistema originalmente è stato proiettato per usare una Base di Dati non relazionale MongoDB per motivi strettamente contrattuali. Per richiesta dell'utente e per motivi di privacy, il database proiettato per questo progetto è stato totalmente cambiato anche se i concetti di business rimangono gli stessi. È stato cambiato anche il tipo di database per uno relazionale, specificamente Oracle DB. Il processo di cambio non ha prodotto un impatto significativo, dato l'uso del framework Spring Data e la creazione di file di configurazioni che permettono di fare questo tipo di migrazioni tecnologiche in modo semplice e ordinato. Dati i requisiti utente si è proiettato un modello relazionale che viene illustrato in modo sintetico di seguito.

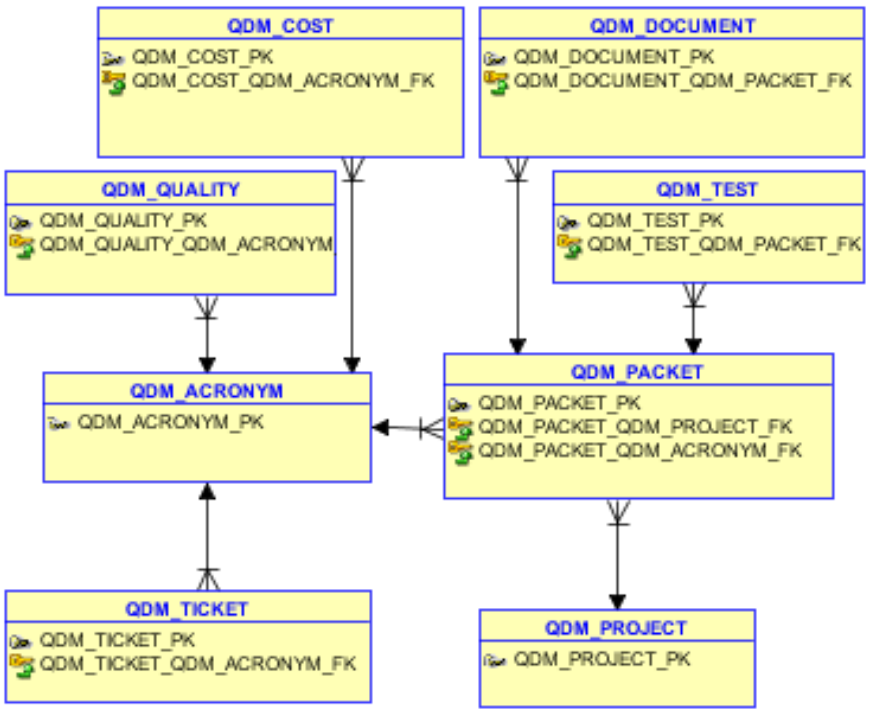

Figura 30: Modello relazione sintetico

Per motivi di spazio questo modello è stato semplificato rappresentando solo i campi chiave. Per vedere il modello completo si può fare riferimento al ***Capitolo 7 Allegati***.

Nel modello architetturale del sistema QDM, il DB si trova dentro un container Docker per lasciare le porte aperte a future implementazioni di concetti come “Data replication”, “Backup-Database”, “Database-Segmentation” e molto altro.

Capitolo 6 Conclusioni ed implementazioni future

Lo scopo di questo progetto era quello di disegnare un prodotto software da proporre ad una grande azienda, come miglioramento del suo processo di ispezione della qualità dei rilasci software in un ampio dominio di dati. La presentazione è avvenuta e lo scopo è stato raggiunto, anche se l'azienda non ha voluto accogliere la proposta. Comunque, grazie ad uno sforzo a livello commerciale e strategico, l'azienda ha disegnato un framework proprio su cui basare tutti i nuovi progetti in ambito web e su cui verranno migrati molti dei progetti già esistenti. L'analisi effettuata dal gruppo di lavoro che ha disegnato e implementato questo framework ha tenuto in considerazione alcune delle idee e impostazioni che sono state disegnate da progetti simili a Quality Data Meter (elaborati anche da Sopra Steria, l'azienda promotrice di questo progetto). La comprensione degli argomenti e gli elementi presenti in questo progetto sono stati di grande utilità per l'azienda proponente a livello di know-how, e ancora oggi si continua a cercare di approfondire e diffondere il più possibile.

6.1 Conclusioni

Quality Data Meter è stato un progetto che ha raggiunto, a livello tecnico, la maggior parte dei suoi obbiettivi. Resta da dire che i punti che rimangono aperti riguardano implementazioni di funzionalità che non si possono definire in un modo approfondito, dato che i requisiti associati fanno parte del funzionamento interno di altri applicativi che non permettono la condivisione dell'informazione necessaria. Le attività concluse fino alla fase di presentazione del progetto sono state adattate al fine di mostrare all'utente una necessità e un prodotto che potesse soddisfarla.

La fase di raccolta di requisiti, analisi della necessità e disegno delle soluzioni è stata fondamentale per capire quale fosse il miglior modo di affrontare questo progetto e quali fossero i punti fondamentali del prodotto che si voleva presentare all'utente. Capire i meccanismi, sistemi e processi coinvolti in queste attività è stato complesso e dispendioso perché i gestori delle informazioni necessarie per lo sviluppo del progetto sono quasi tutti in mano ad aziende di consulenza esterne e non sono sempre disponibili a fornire dettagli del loro lavoro. Inoltre, dopo una grande ricerca di informazione, è stata necessaria una fase di standardizzazione e profilazione dei dati poiché ogni fornitore aveva una diversa comprensione dei dati a seconda dell'ambito in cui lavorava, quindi non esisteva una "naming convention" standard per tutti gli applicativi. Dunque, la fase di analisi dei dati è stata lunga e difficoltosa ma ha portato ad una soluzione efficiente, veloce e facilmente scalabile.

La prosecuzione degli sviluppi di QDM è avvenuta in modo indipendente e totalmente propedeutico alla presentazione di questa tesi. Il progetto esposto in questa tesi si discosta da quello originale per questioni di privacy, ciononostante, al fine di rendere l'applicativo il più simile possibile sia a livello grafico che logico a quello reale, i componenti illustrati sono parte integrante del progetto originale.

Le tecnologie, i framework e i tool usati sono stati di grande supporto per lo sviluppo e la distribuzione dell'applicativo; si considera che abbiano portato vantaggi e, date le caratteristiche trovate nel prodotto finale, è possibile dire che in termini di proprietà del software i requisiti sono stati totalmente soddisfatti.

6.2 Sviluppi futuri

Data l'architettura proiettata, l'espansione dell'applicativo QDM ha molti orizzonti, ovvero ci sono molti argomenti che si potrebbero integrare facilmente a questa soluzione. La creazione di nuovi widget e la creazione di nuovi servizi sono facilmente realizzabili, così come la manutenzione del sistema esistente. Accentrando tutte le informazioni riguardanti un'area così vasta come il processo di rilascio del software, è facile pensare che argomenti della stessa area potrebbero essere implementati all'interno del progetto dando loro visibilità e consentendo di accentrare le informazioni.

Specificamente, pensando nello stato del progetto attuale, sarebbero possibili diversi miglioramenti e aggiunte per favorire la rappresentazione dell'informazione nei widget. Prima di tutto si vuole chiarire che la scelta di sviluppare widget diversi è stata fatta pensando alla varietà di fonti di informazione, per cui ogni widget doveva non solo mostrare informazioni diverse ma era anche filtrabile a suo modo.

La pagina di Statistiche ha un potenziale di crescita enorme dato che tutti i dati presentati in questo applicativo fanno riferimento a misure numeriche che permettono la creazione di una grande quantità di grafici per permettere l'analisi dello stato attuale ma anche dei diversi stati attraverso il tempo.

Adesso, avendo una prospettiva più obbiettiva dei widget, sarebbe ideale pensare ad un widget unico, di maggiori dimensioni, che permetta di filtrare per Progetto e per step e che possa mostrare informazioni relative ai tre widget che hanno questo tipo di filtrazione. Questo sarebbe molto utile per i capiprogetto e le persone del quality release monitoring che devono vedere tutte le informazioni per ogni filtro. In questo modo si dovrebbero mantenere i widget attuali dato che i ruoli come quello degli sviluppatori o dei tester dovrebbero avere anche una visione accentrata ad un argomento unico relativo al loro lavoro.

In ambito dei widget, ci sarebbe da proporre un'implementazione di reportistica basata su fogli di calcolo con tabelle contenenti i dati filtrati e raggruppati provenienti da ogni widget. I pulsanti sono stati aggiunti al modello visuale in modo simulato ma funzionalmente non è stata fatta nessuna elaborazione. Questi hanno l'aspetto mostrato a continuazione.

Figura 31: Pulsante esportazione Excel

Oltre a tutte le idee presentate, ci sarebbe da implementare un sistema di profilazione utente per permettere a ruoli diversi coni di visibilità più adeguati alle loro funzioni. Ad esempio, gli sviluppatori avrebbero più interesse nel vedere i widget attinenti alla qualità del software e alle scadenze relative al loro rilascio, mentre i tester avrebbero bisogno di vedere quelli relativi a tutta la fase di test, manuali e automatici. Inoltre, sarebbe utile presentare, per questo tipo di utenze, solo i progetti associati ai loro profili; per il gruppo di quality release monitoring, invece, sarebbe utile una visione panoramica di tutti i dati.

6.2.1 Esempio

Dato il sistema attuale, per spiegare il processo implementativo di nuovi segmenti applicativi verrà fornito un esempio pratico: si vuole aggiungere un nuovo flusso dati di input proveniente da un sistema alimentante nuovo per popolare un widget che dia evidenza dell'indice di qualità statica del codice, associato alle nuove righe implementate.

Per l'acquisizione dei dati ci sono due opzioni diverse. La prima sarebbe l'implementazione di un nuovo parser all'interno del modulo Engine esistente per permettere l'acquisizione e il processamento dei dati. Questo implica estendere la funzionalità attuale del progetto e, a livello architetturale, l'aggiornamento dei container associati. La seconda opzione sarebbe l'implementazione di un nuovo modulo Engine associato alla funzionalità che verrà distribuito in altri container per permettere il flusso dei dati verso il sistema. A livello architetturale implica creare nuovi container che possono lavorare in parallelo a quelli già esistenti ma ha lo svantaggio di dover usare molte più risorse per la loro distribuzione e funzionamento.

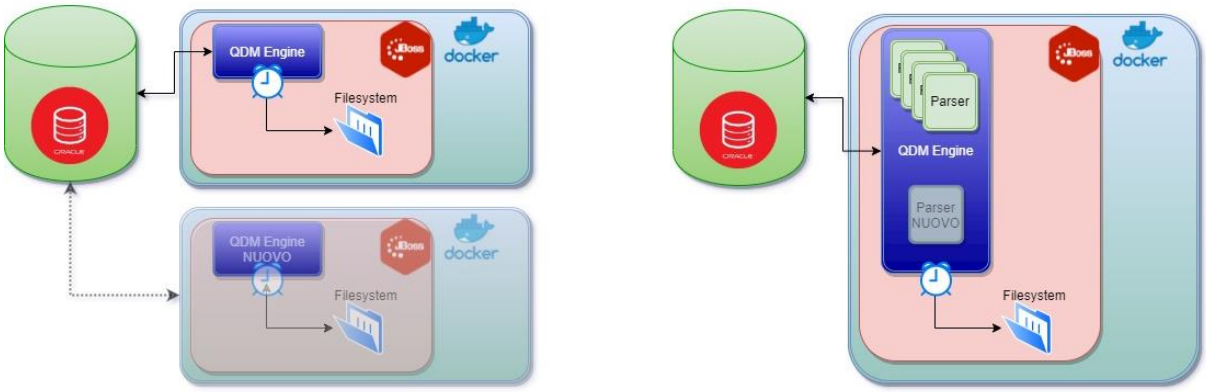

Figura 32: Modulo Engine, nuova funzionalità

La decisione deve essere presa tenendo in conto principalmente la quantità di dati che si prevedono per l'elaborazione e le risorse computazionali del Host.

Invece per la parte Services, ci sarebbe un nuovo modulo contenente tutti i servizi di cui il nuovo argomento avrebbe bisogno dato l'isolamento previsto nel progetto a livello di tematica. Questo implica che, rispettando la progettazione disegnata, l'unica soluzione possibile è quella illustrata nella Figura 33.

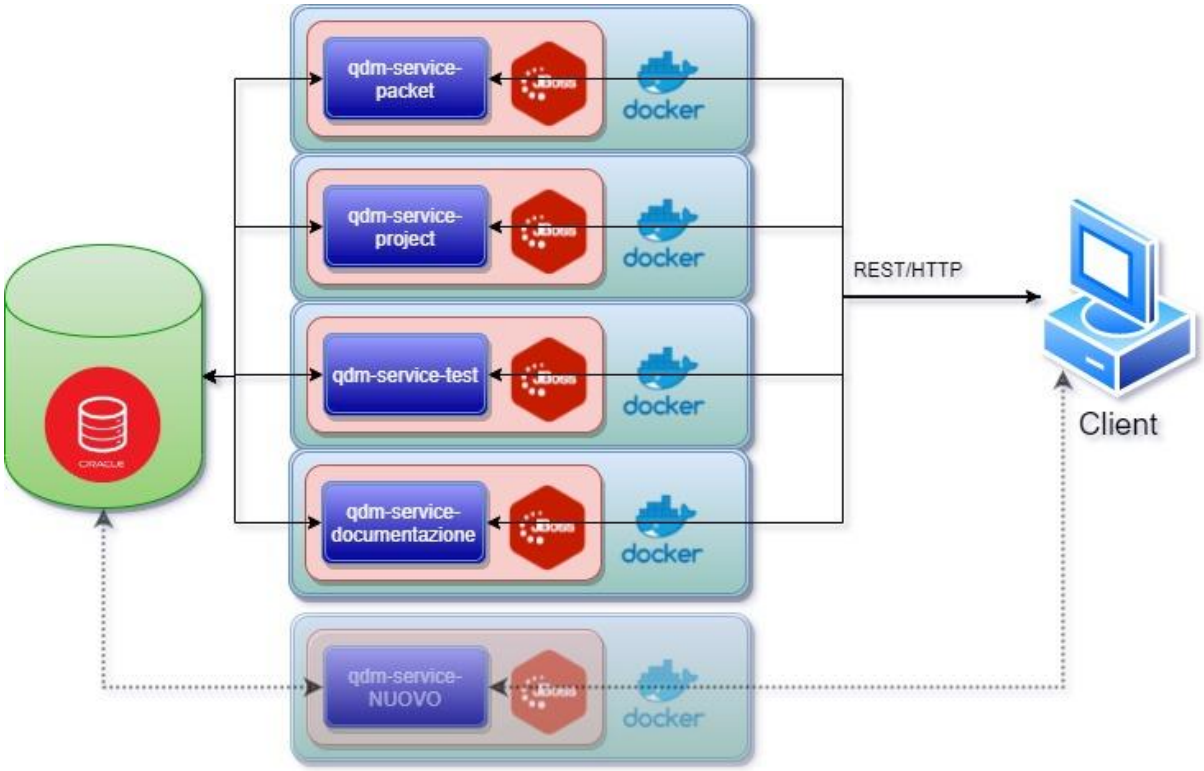

Figura 33: Moduli Services, nuova funzionalità

A livello Front-end, il lavoro è più semplice dato che la maschera del widget esiste già e deve essere soltanto popolata con i componenti grafici utili alla selezione e comprensione dei dati nel miglior modo possibile. Inoltre, ci sarebbe bisogno di aggiungere una nuova chiamata ai nuovi servizi nel Call Service, e il passaggio dei dati dal gestore Data Service. In questo modo i componenti relativi al widget sarebbero collegati a tutte le informazioni disponibili dal Back-End.

Capitolo 7 Bibliografia

Barroca, L., Hall, J., & Hall, P. (2000). *Software Architectures.* Londra: Springer.

Comai, A. (2006). *LINEE GUIDA - UML - MODELLI ARCHITETTURALI.*

Fowler, M. (2014, Marzo 25). *Microservices*. Tratto da martinFowler.com: https://martinfowler.com/articles/microservices.html

Fowler, M. (2019, 08 01). *Software Architecture Guide*. Tratto da martinFowler.com: https://martinfowler.com/architecture/

Google. (s.d.). *Angular guide*. Tratto da Angular: https://angular.io/guide/what-is-angular

Google. (s.d.). *Angular guide - Architecture-modules*. Tratto da Angular: https://angular.io/guide/architecture-modules

Google. (s.d.). *Angular guide - Component*. Tratto da Angular: https://angular.io/guide/component-overview

Huß, R. (2015 - 2020). *fabric8io/docker-maven-plugin*. Tratto da docker-maven-plugin: https://dmp.fabric8.io/

JBoss; Docker. (s.d.). *Docker-Hub - jboss/wildfly*. Tratto da Docker-Hub: https://hub.docker.com/r/jboss/wildfly/

Oracle. (s.d.). *Java - Che cos'è la tecnologia Java e a cosa serve?* Tratto da Java: https://www.java.com/it/download/help/whatis_java.html

Oracle; Docker. (s.d.). *Docker-Hub - Oracle Database Enterprise Edition*. Tratto da Docker-Hub: https://hub.docker.com/_/oracle-database-enterprise-edition

RedHat. (s.d.). *RedHat*. Tratto da Microservizi: https://www.redhat.com/it/topics/microservices/what-are-microservices

Spring. (2021). *Spring Framework*. Tratto da Spring: https://spring.io/

The Apache Software Foundation. (2002-2021). *Apache Maven Project*. Tratto da Apache Maven: https://maven.apache.org/

Thomas, D., & Hunt, A. (2020). Pragmatic Programmer. Apogeo.