



# UNIVERSITÀ DI PARMA

---

Dipartimento di Ingegneria e Architettura  
Corso di Laurea in Ingegneria Informatica, Elettronica e delle  
Telecomunicazioni

## Reingegnerizzazione di una Dashboard di Business Intelligence

Reengineering of a Business Intelligence Dashboard

Relatore:  
Chiar.mo Prof. Michele Amoretti

Correlatore:  
Dott. Antonio Calò

Tesi di Laurea di:  
Leonardo Minaudo

---

ANNO ACCADEMICO 2021-2022

Ai miei genitori e a mia sorella.



# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Big Data e Business Intelligence</b>	<b>3</b>
1.1 Big Data . . . . .	3
1.1.1 Big Data e Pubbliche Amministrazioni (PA) . . . . .	4
1.2 Business Intelligence . . . . .	5
1.3 Applicativo Gzoom . . . . .	8
1.3.1 Architettura . . . . .	8
1.3.2 Modularità . . . . .	13
1.3.3 Interoperabilità . . . . .	14
1.3.4 Algoritmi per il Calcolo dei Risultati . . . . .	16
<b>2 Descrizione Funzionale</b>	<b>20</b>
2.1 Reingegnerizzazione . . . . .	20
2.1.1 Microservizi . . . . .	21
2.1.2 API REST . . . . .	22
2.1.3 Single Page Application . . . . .	26
2.2 Dashboard di Business Intelligence . . . . .	28
2.2.1 Funzionalità della Nuova Dashboard . . . . .	29
Use Case Diagram . . . . .	30
Descrizione Funzionale . . . . .	31
Struttura dei servizi e gestione delle richieste/risposte	36

---

<b>3</b>	<b>Implementazione</b>	<b>39</b>
3.1	Backend . . . . .	39
3.1.1	Microservizi - Spring Boot . . . . .	39
3.1.2	Query . . . . .	44
	QueryDSL . . . . .	44
	MyBatis . . . . .	45
3.2	Frontend . . . . .	48
3.2.1	Angular . . . . .	48
3.2.2	Tachimetri - Ngx-gauge . . . . .	52
3.2.3	Grafici - Chart.js . . . . .	54
<b>4</b>	<b>Risultati</b>	<b>57</b>
	<b>Conclusioni</b>	<b>66</b>
	<b>Bibliografia</b>	<b>69</b>

# Introduzione

In un settore in continua evoluzione l'obsolescenza del software è rapida e aumenta i costi di manutenzione. Per risolvere tale problema si ricorre alla reingegnerizzazione del software, processo utile a trasformare e migliorare un'architettura software già esistente, permettendo di guadagnarne in prestazioni e scalabilità.

Questa Tesi è frutto di un tirocinio presso l'azienda *Maps Group* [1] che sviluppa soluzioni tecnologiche che trasformano i dati disomogenei e disaggregati in una base di informazioni, e dunque di conoscenza, messa al servizio delle scelte di Operational Business Intelligence. Il tirocinio, che è stato il punto di partenza per questa Tesi, ha avuto come obiettivo la reingegnerizzazione della dashboard di Business Intelligence dell'applicativo Gzoom, facente parte delle diverse soluzioni che l'azienda Maps Group propone. Nello specifico l'attività svolta ha avuto il compito di migliorare l'architettura e fornire nuove funzionalità alla dashboard, contribuendo a un processo di rimodernizzazione dell'intero applicativo. Per farlo è stato utile capire il dominio applicativo in cui la dashboard opera cioè l'elaborazione e l'analisi dei dati.

La Tesi è suddivisa in quattro capitoli. Il Capitolo 1 ha il compito di descrivere l'importanza dei dati, cosa sono i Big Data e cos'è la Business Intelligence, descrivendo il processo tecnologico che elabora i dati grezzi e ne estrae informazioni utili. Sempre nel Capitolo 1 viene presentata l'architettura, la

modularità, l'interoperabilità e gli algoritmi per il calcolo dei risultati dell'applicativo Gzoom. Nel Capitolo 2 vengono esposti i motivi della reingegnerizzazione e le soluzioni architetturali scelte per farlo, insieme a una descrizione funzionale della dashboard sviluppata. Con il Capitolo 3 vengono descritte le tecnologie utilizzate per lo sviluppo della dashboard di Business Intelligence soffermandosi su alcune caratteristiche implementative di tali strumenti che sono state approfondite per la fase progettuale di realizzazione. Infine, nel Capitolo 4, vengono mostrati i risultati ottenuti. Successivamente nelle Conclusioni vengono ripresi quelli che sono stati gli obiettivi di reingegnerizzazione e confrontati con i risultati ottenuti, proponendo, inoltre, possibili sviluppi futuri della dashboard.

# Capitolo 1

## Big Data e Business Intelligence

### 1.1 Big Data

Nell'era tecnologica in cui stiamo vivendo la maggior parte delle persone si interfaccia con sistemi informativi i quali, oltre ad intrattenere, facilitano e agevolano lo svolgimento di molteplici attività, generando dati che possono essere immagazzinati e elaborati. Tali dati sono in continua crescita soprattutto con la progressiva trasformazione digitale di aziende e pubbliche amministrazioni, così generando una grande mole di informazioni da gestire. Come si sente spesso parlare i dati vengono definiti il nuovo petrolio proprio per il valore che deriva dalla loro elaborazione.

Il termine **Big Data** viene utilizzato per far riferimento alle grandi masse di dati caratterizzati dalle 3 V:

- **Volume:** i Big Data sono un insieme di molteplici dati, che possono pesare sull'ordine di grandezza di terabyte o petabyte;
- **Varietà:** i formati contenenti dati possono variare, ad esempio in documenti txt, PDF, Word, Excel, ecc., quindi non per forza rappresentati sotto forma tabellare in un database relazionale. Inoltre, i dati possono essere generati automaticamente da macchine, da utenti o da altre fonti;



- **Velocità**: è importante rendere disponibili i nuovi dati il prima possibile tramite strumenti che garantiscono il corretto immagazzinamento.

I Big Data sono anche caratterizzati da **variabilità**, poiché soggetti spesso al cambiamento e alla variazione continua. La loro **complessità** aumenta al crescere delle dimensioni di un dataset ma il vero valore si calcola in **veridicità** che si riferisce alla qualità dei dati e al valore informativo che può essere estratto da essi [2].

I dati utilizzati in maniera opportuna possono aiutare aziende e enti pubblici per, ad esempio:

- Prendere decisioni;
- Sviluppare nuovi prodotti;
- Ottimizzare offerte o servizi;
- Ridurre i tempi;
- Tagliare i costi.

Punti di fondamentale importanza per la crescita tempestiva di un'azienda ma anche della Pubblica Amministrazione che potrebbe offrire anche una maggiore tutela e trasparenza in termini di finanze e privacy.

### 1.1.1 Big Data e Pubbliche Amministrazioni (PA)

La PA sfrutta anche gli **Open Data**, dati accessibili liberamente, che permettono ai cittadini di rimanere informati dei processi decisionali dell'ente. Si differiscono dai Big Data poiché quest'ultimi non debbono essere per forza accessibili e pubblici.

"I dati delle pubbliche amministrazioni sono formati, raccolti, conservati, resi disponibili e accessibili con l'uso delle tecnologie dell'informazione e della comunicazione che ne consentano la fruizione e riutilizzazione, alle condizioni fissate dall'ordinamento, da parte delle altre pubbliche amministrazioni e dai privati" (art. 50, comma 1)

Ricorrendo a sistemi informativi in grado di analizzare i Big Data una Pubblica Amministrazione ha il vantaggio di poter gestire meglio i servizi pubblici:

- Diffondendo più facilmente informazioni verso cittadini e imprese;
- Standardizzando l'utilizzo degli Open Data minimizzando i costi per l'accesso e l'utilizzo dei dati;
- Analizzando e gestendo i dati direttamente all'interno della Pubblica Amministrazione, in modo da aumentare la consapevolezza dei dati a disposizione e creare servizi "intelligenti" e utili alla comunità [3].

## 1.2 Business Intelligence

Ad oggi si traggono molti benefici dall'analisi dei dati soprattutto dalla loro elaborazione per poterne trarre informazioni. Informazioni che possono fungere come mezzo di supporto per il processo decisionale, per esempio, di un'azienda o di pubbliche amministrazioni.

La **Business Intelligence (BI)** racchiude tutti gli strumenti e le tecniche utilizzati per convertire i dati grezzi in informazioni utili. È un processo tecnologico che mette insieme business analytics, data mining, visualizzazione dei dati, strumenti e infrastrutture per i dati, tutte best practice per aiutare le organizzazioni a prendere decisioni basate sui dati.

Gli step fondamentali su cui si basa la Business Intelligence sono:

1. Raccolta dei dati;
2. Analisi dei dati;
3. Reporting;
4. Monitoraggio e previsione.

La circolarità del processo di Business Intelligence fa sì che nell'ultimo step di monitoraggio e previsione si possa tornare alla raccolta dei dati.

La BI si basa sui Big Data. Per avere a disposizione una mole di dati bisogna recuperarli da fonti già esistenti o raccoglierli tramite l'uso di sistemi informativi in grado di fornire un'interfaccia utente che permetta agilmente di inserire i dati. Successivamente si passa all'**analisi dei dati** per trasformarli da grezzi a informazioni utili. I tre tipi di analisi più comuni sono:

- **Analisi del foglio di calcolo:** i dati da un foglio di calcolo vengono tradotti in grafici e tabelle;
- **Utilizzo di query di dati:** i dati vengono analizzati dal software dopo l'importazione;
- **Strumenti di visualizzazione:** i dati vengono visualizzati, senza subire nessuna elaborazione, tramite grafici e diagrammi in modo tale che gli utenti possano leggerli e avere un quadro generale di quello che rappresentino.

Dopo l'analisi si passa al **reporting**, le informazioni devono essere sistemate in dei resoconti, prospetti utilizzabili per un confronto tra target e consuntivi, dove i target rappresentano l'obiettivo da raggiungere e il consuntivo indica il risultato raggiunto in quel momento. Nella fase di monitoraggio l'end user può ispezionare dati e informazioni tramite:

- **Dashboard:** insieme di rappresentazioni grafiche per agevolare la lettura dei dati utili agli utenti;
- **Key Performance Indicators (KPI):** indicatori di performance utilizzati per valutare le prestazioni complessive a lungo termine;
- **Business Performance Management:** processo che permette di attuare obiettivi e misure applicabili a livello operativo. In tal campo rientra la previsione che appoggia il management a fare previsioni su

cosa accadrà sulla base dei dati disponibili e tendenze raccolti durante le fasi di analisi e monitoraggio, in modo da suggerire mosse future o quali dati analizzare in futuro.

La previsione può essere suddivisa in due tipologie: **data mining** e **modellazione predittiva**. Il data mining si occupa di trovare modelli e relazioni in e tra grandi insiemi di dati, in modo da poter estrarre o trasformare i dati in modo da poterli utilizzare e capire ulteriormente. La modellazione predittiva invece mira a prevedere il risultato di un'azione o la probabilità di un risultato [4].

Lo scopo della Business Intelligence è di permettere alle organizzazioni di fissare degli obiettivi e aiutarli a trovare la strada migliore per arrivare al loro completamento tramite delle domande che possono riguardare sia il passato che il futuro. Nel tempo però sono cambiate le modalità per arrivare a tal fine.

Nella Business Intelligence tradizionale venivano utilizzati report statici per la quasi totalità delle domande di analisi. Questo comportava dei cicli di report lenti che non permettevano immediatezza nelle risposte, grosso svantaggio quando bisognava prendere decisioni immediate. Nonostante questo approccio non sia stato completamente abbandonato oggi viene applicato un processo dinamico e interattivo che rende fruibili i dati anche a chi non ha particolari competenze di IT<sup>1</sup>.

In ambito tecnologico i software che trattano di BI sono progettati per acquisire grandi quantità di dati per tradurli in report ed applicare statistiche in forma grafica, comprensibile da qualsiasi utente. Gestiscono i Big Data in tempo reale garantendone l'analisi e fornendo risultati in tempi rapidi [6].

---

<sup>1</sup>*Information Technology*: insieme dei metodi e delle tecnologie che vengono utilizzate in ambito pubblico, privato o aziendale per l'archiviazione, la trasmissione e l'elaborazione di dati e informazioni [5].

## 1.3 Applicativo Gzoom

**Gzoom** [7] è un software gestionale open source creato per migliorare risultati, gestire le risorse umane e tenere sotto controllo i risultati ottenuti da amministrazioni, enti e aziende. Tratta di dati e di come poterne creare un valore pubblico aggiuntivo, fornendo aiuto in campi come la trasparenza, anticorruzione, privacy, gestione dei premi aziendali e del modello organizzativo previsto dalla 231<sup>2</sup>. Gzoom analizza le prestazioni secondo una **pianificazione strategico-operativa** e in base ai budget. L'applicativo mette in risalto gli aspetti economici e i processi organizzativi, per la generazione di dati che aiutino a compiere scelte di management implementando metodologie di Business Intelligence. Inoltre, permette di tenere sotto controllo il livello di efficienza, le performance individuali, la tracciabilità degli obiettivi che si vogliono raggiungere e l'effettivo avanzamento dei progetti [9].

### 1.3.1 Architettura

L'architettura di Gzoom è basata su **microservizi** che dialogano tra loro tramite **API**<sup>3</sup>, in modo tale da accelerare il time-to-market<sup>4</sup> delle nuove funzionalità. **GzoomLegacy** costituisce il servizio principale che si basa su **OFBiz (Open For Business)**, progetto open source dell'Apache Software Foundation che consiste in una suite di applicazioni aziendali sufficientemente flessibile da poter essere utilizzata in qualsiasi settore. L'architettura comu-

---

<sup>2</sup>Un Modello Organizzativo e di Gestione ai sensi del DLgs 231/2001 è un insieme di protocolli, che regolano e definiscono la struttura aziendale e la gestione dei suoi processi sensibili. Il Modello Organizzativo 231, se correttamente applicato, riduce il rischio di commissione di illeciti penali [8].

<sup>3</sup>*Application Programming Interfaces*: insieme di protocolli che permettono di implementare servizi di varia natura e farli interoperare senza che essi debano preoccuparsi della comunicazione, semplificando così lo sviluppo delle app e consentendo un netto risparmio di tempo e denaro.

<sup>4</sup>Periodo di tempo che intercorre tra l'ideazione di un prodotto e la sua effettiva commercializzazione.

ne di tale framework consente agli sviluppatori di estenderla e/o migliorarla facilmente per creare funzionalità personalizzate [10].

GzoomLegacy non lavora da solo ma è affiancato dai servizi **Gzoom2Backend** e **Gzoom2ReportBackend**, applicazioni sviluppate in Java con il framework **Spring Boot**, che utilizzano lo stesso modello dati del servizio principale.

L'insieme di questi tre servizi forma una piattaforma software integrata e completa per la costruzione di servizi e di applicazioni web, che consente di costruire soluzioni basate su interfacce web ricche di funzionalità evolute e permette inoltre una facile integrazione con le banche dati e altri servizi web. La piattaforma comune fornisce un insieme di funzionalità di base necessarie alla maggior parte delle applicazioni Web evolute (come l'accesso a Database, disponibilità di librerie di componenti visuali, linguaggi di scripting, interfaccianti SOAP, XML, CSV, PDF) e standardizza le modalità di costruzione delle applicazioni stesse, in modo da ridurre il costo di manutenzione dell'intera piattaforma.

L'applicativo implementa funzioni utente per la configurazione del sistema, funzioni di processo per la definizione, misurazione e valutazione degli obiettivi attraverso gli indicatori ad essi associati, nonché il processo di produzione dei risultati con la creazione di reports e cruscotti. Gzoom estende le funzionalità di OFBiz e ne aggiunge delle personalizzazioni con componenti dedicati non dipendenti dal framework. In merito all'erogazione dei servizi nel fornire informazioni fruibili, senza discriminazioni, anche per utenti che hanno bisogno di configurazioni particolari (es. per persone ipovedenti) vengono usati diversi temi grafici.

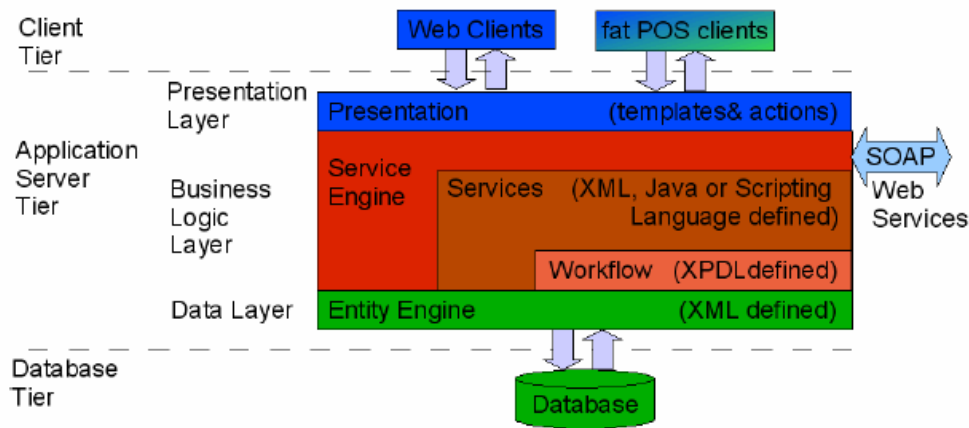


Figura 1.1: Struttura di OFBiz

OFBiz, essendo un framework, fornisce delle regole di sviluppo che devono essere rispettate per far sì che non vengano meno gli standard qualitativi. Come si vede in Figura 1.1 è strutturato secondo un'architettura web a tre Tier formata da componenti/moduli ciascuno con la propria interfaccia grafica, logica di business, entità e definizione di servizi web secondo una struttura **SOA (Service Oriented Architecture)** comunicanti tramite il protocollo **SOAP (Simple Object Access Protocol)**.

**Gzoom2Backend** e **Gzoom2ReportBackend** sono basati su un'architettura a servizi che si interfacciano tramite API **REST (Representational State Transfer)**, fanno parte del componente identificato come **WebApp** nel component diagram in Figura 1.2.

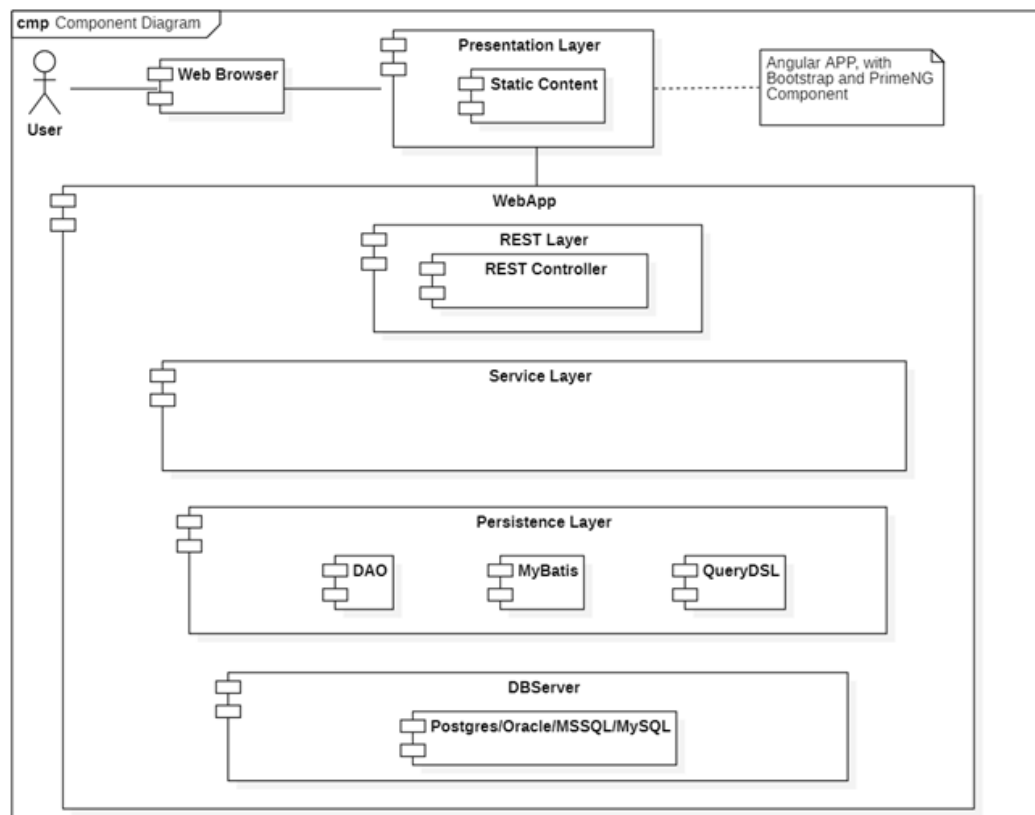


Figura 1.2: Component Diagram Gzoom2

La WebApp è sua volta formata dai seguenti sottocomponenti:

- **REST Layer:** contiene i controller in grado di gestire tutte le richieste che arrivano dai client;
- **Service Layer:** definisce il limite di un'applicazione e il suo insieme di operazioni disponibili dal punto di vista dell'interfacciamento dei livelli client. Incapsula la logica di business dell'applicazione, controllando le transazioni e coordinando le risposte nell'implementazione delle sue operazioni [11];
- **Persistence Layer:** fanno parte di questo livello il **DAO (Data Access Object)**, il framework **myBatis** e l'API **QueryDSL**. Tale livello contiene tutta la logica di archiviazione e traduce gli oggetti di business



da e verso i record del database [12]. QueryDSL consente di oltrepassare la mancata corrispondenza tra l'approccio orientato agli oggetti e il modello di database relazionale, gestendo tutte le operazioni CRUD agevolmente. MyBatis con la sua flessibilità permette di gestire le query più complesse;

- **DBServer**: nel livello database vengono eseguite le operazioni CRUD (create, read, update, delete) [12].

Il componente WebApp si interfaccia con l'utente tramite il **Presentation Layer**, secondo una struttura **Model-View-Controller (MVC)** in modo da creare indipendenza tra le varie componenti, la gestione di tale livello è delegata al **Gzoom2Frontend**. Al livello di presentazione viene utilizzato il framework **Angular**, per la creazione di applicazioni single-page, e l'insieme di alcune librerie grafiche come **Bootstrap** e **PrimeNG**. Angular permette di sviluppare applicazioni che vengono eseguite interamente lato client alleggerendo il web server, facilita lo sviluppo ed è compatibile con la maggior parte dei web browser moderni (Chrome, Microsoft Edge, Opera, Firefox, Safari e altri). Le librerie grafiche PrimeNG e Bootstrap supportano la creazione di un'interfaccia omogenea con l'utilizzo di componenti UI sviluppati per Angular, come tabelle, menù, card, ecc..

L'utente effettua richieste al sistema accedendo tramite **Web Browser** alla dashboard dell'applicativo. Ogni richiesta viene gestita schematicamente secondo lo schema mostrato in Figura 1.3.

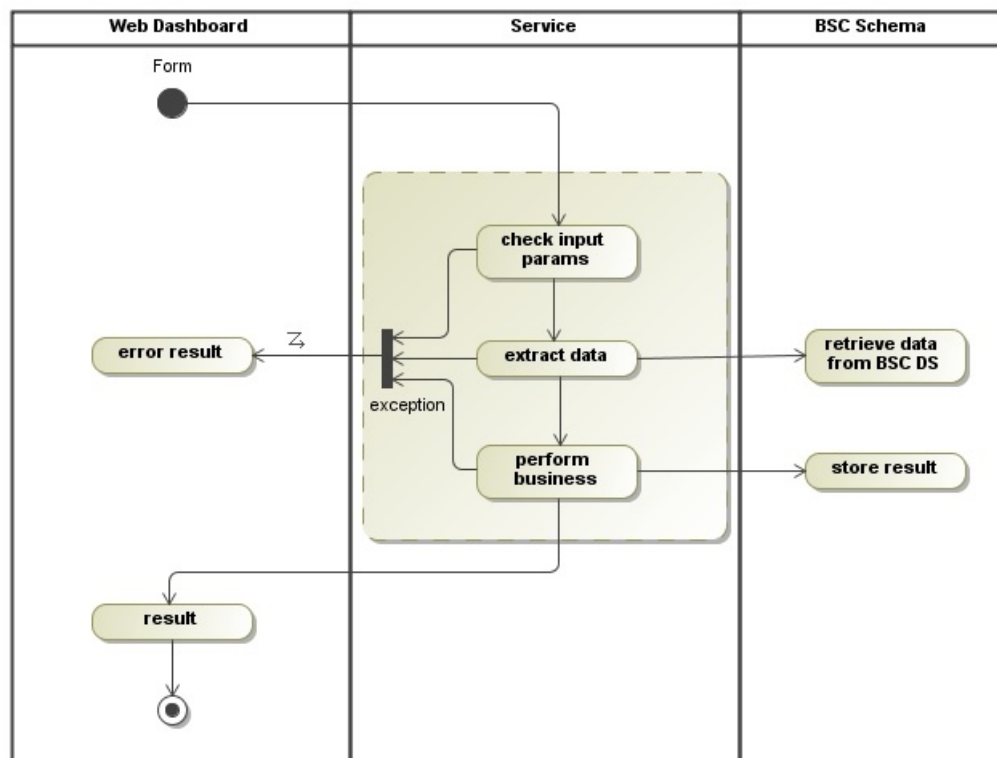


Figura 1.3: Schema per la gestione delle richieste

Inoltrata la richiesta, il servizio ne analizza i parametri di input se presenti e successivamente effettua le operazioni sui dati richiesti e ne fornisce il risultato dell'operazione all'utente, anche in caso di errore.

### 1.3.2 Modularità

Il pattern MVC, insieme agli strumenti e le architetture messe a disposizione da OFBiz, consente maggiore rapidità di sviluppo e rilascio di nuove funzionalità con minore sforzo nel mantenere le esistenti.

Le business logic sono sviluppate come servizi riutilizzabili, scritti in diversi linguaggi secondo la loro applicazione. Ogni servizio svolge un compito preciso ed è indipendente dagli altri, cosicché il servizio grazie alla sua indipen-

denza può essere chiamato sia internamente che esternamente all'applicazione senza conoscere le caratteristiche del sistema chiamante. Tale approccio permette di volta in volta di poter creare servizi applicativi differenti in base a delle esigenze specifiche.

### 1.3.3 Interoperabilità

Gzoom permette l'esportazione completa in XML dei dati presenti nel database e la possibilità di riadattarli in modo da avere un formato per gli Open Data secondo gli standard definiti dal cliente, supportando le API della piattaforma CKAN<sup>5</sup>. Per l'inserimento dei dati il sistema possiede delle tabelle di interfaccia documentate che possono essere popolate con:

- altri applicativi;
- tramite funzione Extract/Transform/Load (ETL) di caricamento presente su Gzoom, che legge in input viste e richiama servizi.

Successivamente i servizi applicativi di Gzoom analizzano le tabelle di interfaccia che sono state popolate e aggiornano le tabelle di business, registrando ogni operazione eseguita in modo tale da poter tenere traccia di eventuali errori riscontrati. Inoltre, è possibile analizzare, tramite servizio, un file Excel normalizzato cosicché da estrarne il contenuto e popolare la tabella di interfaccia standard per poi effettuare il servizio di salvataggio/aggiornamento dati.

---

<sup>5</sup> *Comprehensive Knowledge Archive Network*: è un DMS (Data Management System) open source di riferimento per gli Open Data, utilizzata da governi, organizzazioni e comunità in tutto il mondo, semplifica la pubblicazione, la condivisione e l'utilizzo dei dati [13].

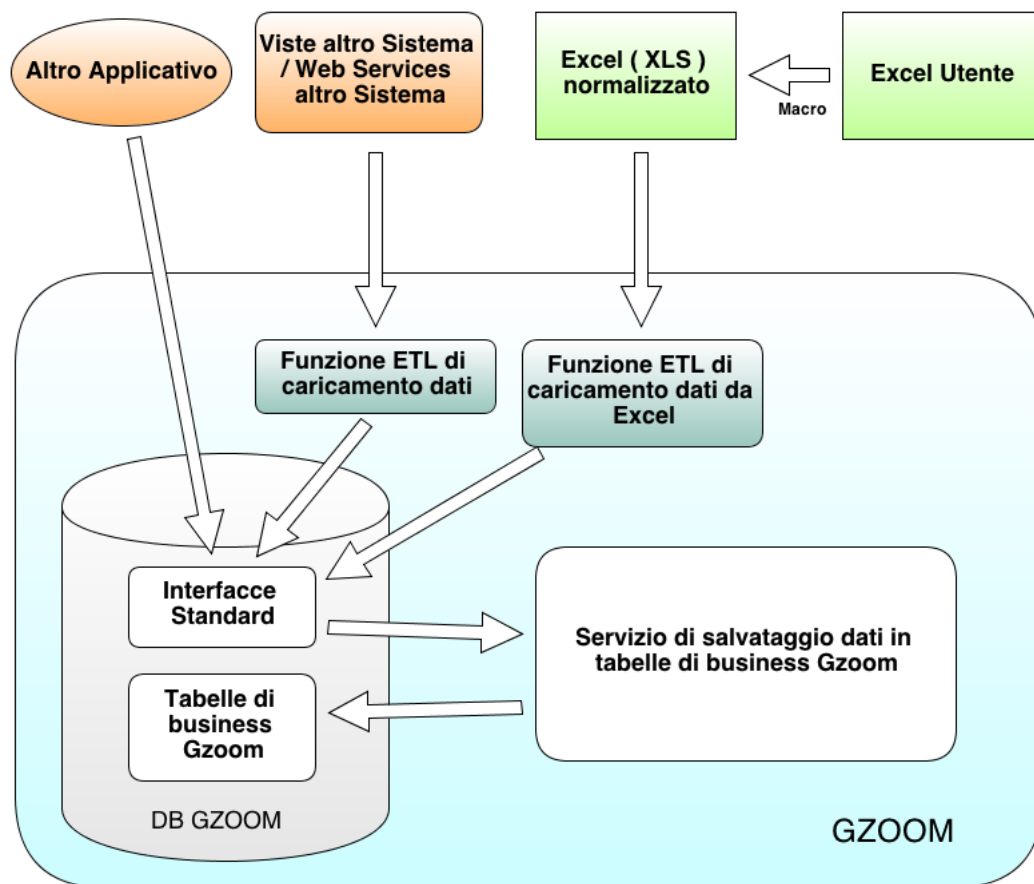


Figura 1.4: Interoperabilità

Segue una breve descrizione degli altri componenti di sistema (Figura 1.4):

- **Altro Applicativo:** rappresenta un applicativo esterno a Gzoom;
- **Viste altro Sistema/Web Services:** viste informative provenienti da banche di dati esterne all'applicativo;
- **Excel (XLS) normalizzato:** servizio che permette la lettura e caricamento di un file Excel (XLS) rispettando l'impostazione della tabella di interfaccia;
- **Excel Utente:** plug-in Excel utilizzato per riportare su strutture normalizzate le entità da interfacciare (indicatori, movimenti e obiettivi);

- **Interfacce Standard:** tabelle di interfaccia utili a standardizzare l'ingresso dei dati;
- **Tabelle di business Gzoom:** tabelle del sistema utilizzate dai servizi di business e per le funzionalità di gestione.

### 1.3.4 Algoritmi per il Calcolo dei Risultati

Gzoom permette alle amministrazioni di tracciare gli obiettivi e i risultati raggiunti tramite l'inserimento e l'avanzamento dei progetti. Per ogni obiettivo viene attribuito un punteggio e degli indicatori di performance (KPI) secondo quanto schematizzato nei seguenti activity diagram:

1. Calcolo del punteggio complessivo (Figura 1.5):  
Tale algoritmo, partendo da un obiettivo identificato come Work Effort (WE), inizializza e cancella i suoi risultati precedenti ed affettua il calcolo del punteggio degli obiettivi figli fino ad arrivare agli obiettivi foglia della gerarchia, per poi determinare il punteggio totale del WE iniziale;
2. Calcolo del punteggio del singolo obiettivo (Figura 1.6):  
Ogni singolo obiettivo viene valutato in base al proprio peso e punteggio KPI insieme ai punteggi degli obiettivi associati;
3. Calcolo degli indicatori di performance KPI (Figura 1.7):  
Il calcolo dei KPI avviene tramite budget consuntivi che misurano dati, informazioni e prestazioni di un periodo passato e il target che rappresenta il raggiungimento dell'obiettivo.

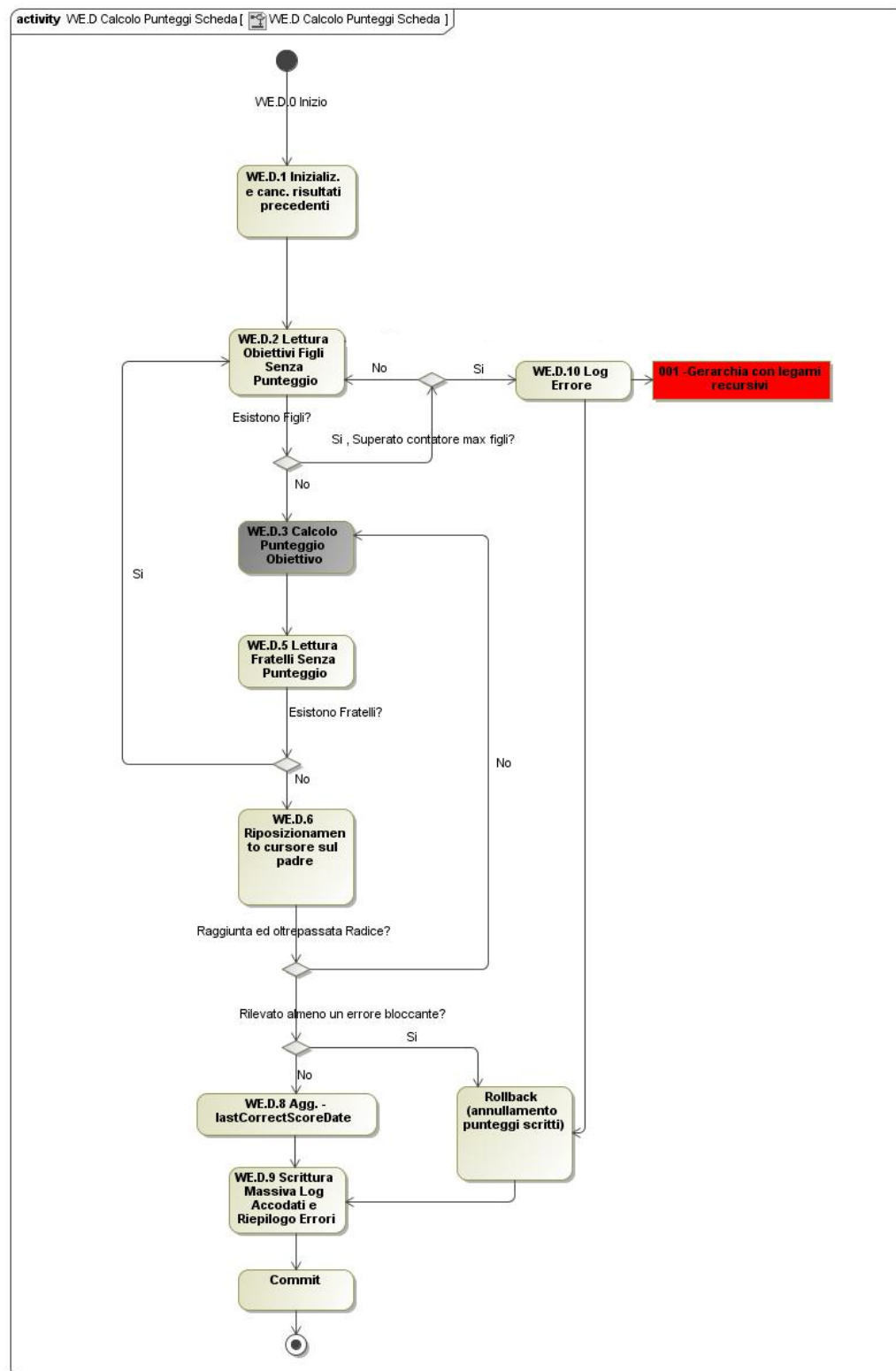


Figura 1.5: Algoritmo per il calcolo del punteggio complessivo

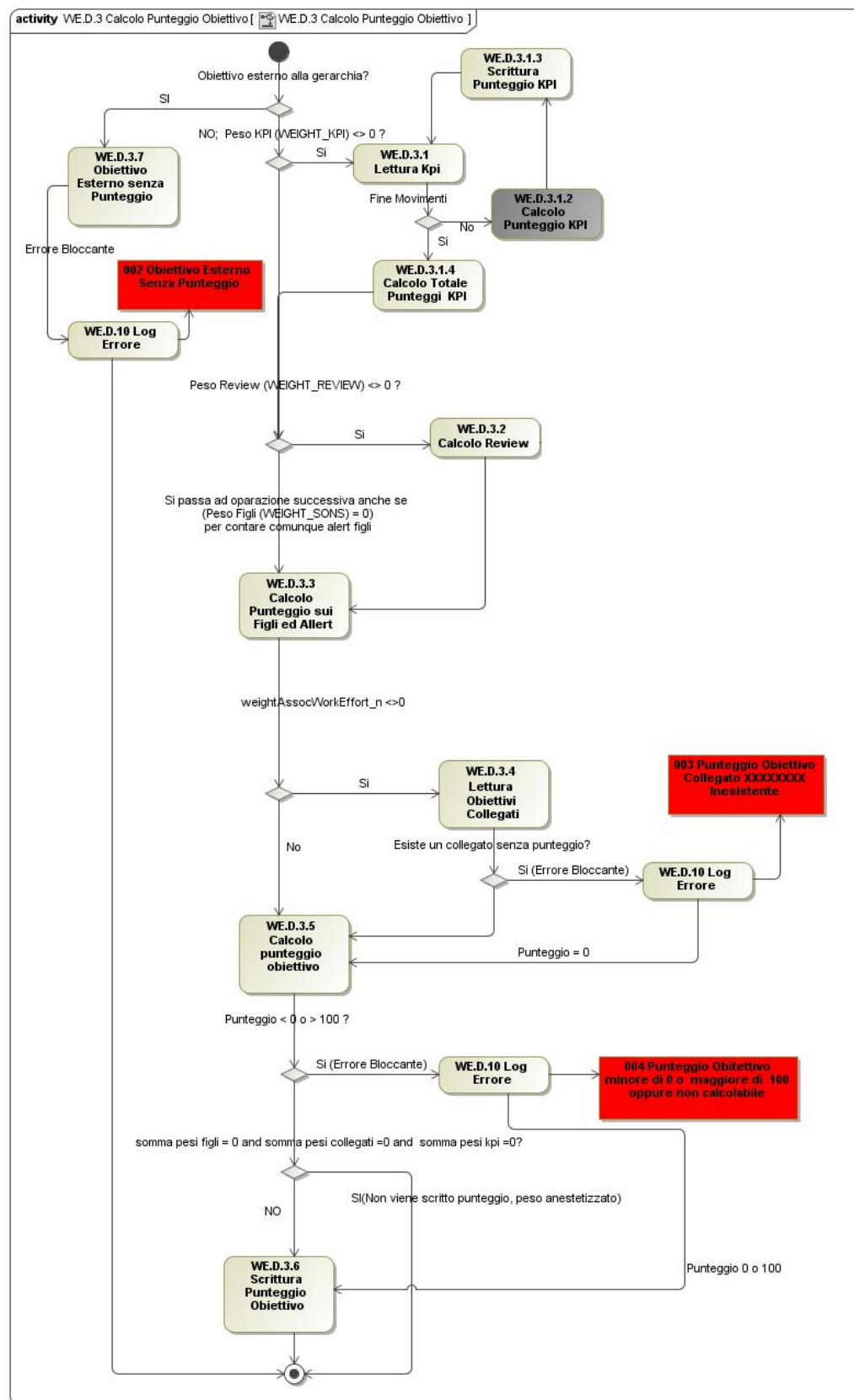


Figura 1.6: Algoritmo per il calcolo del punteggio del singolo obiettivo

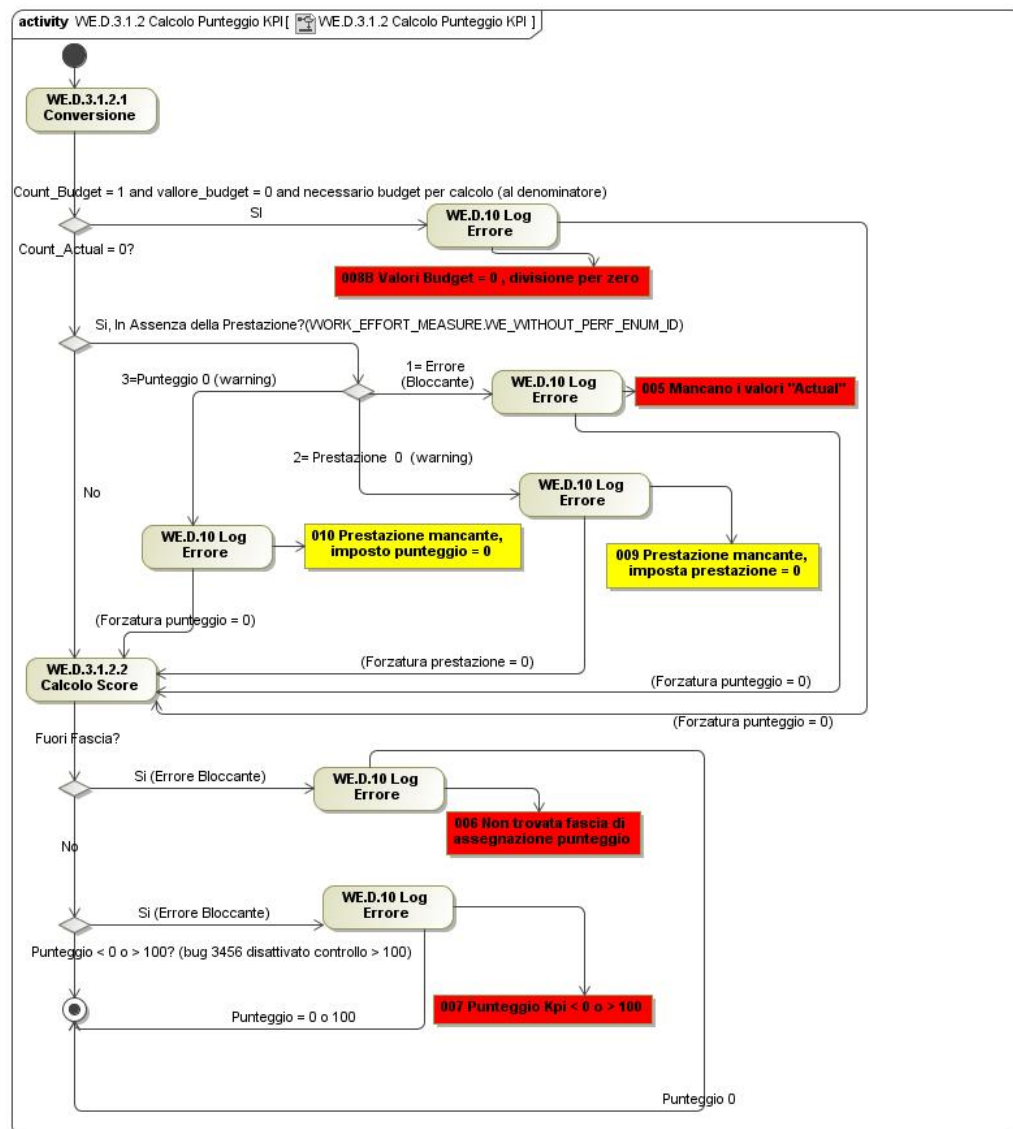


Figura 1.7: Algoritmo per il calcolo degli indicatori di performance KPI



## Capitolo 2

# Descrizione Funzionale

Raccogliere i dati, analizzarli, trarne valore dalla loro elaborazione sono tutti step fondamentali per la buona riuscita di un applicativo di Business Intelligence in cui enti e aziende possono monitorare i propri obiettivi, il loro avanzamento e prendere decisioni di conseguenza. Per tale scopo risulta fondamentale l'utilizzo di dashboard complete in grado di facilitare la lettura dei dati raccolti e delle informazioni elaborate.

### 2.1 Reingegnerizzazione

L'applicativo Gzoom possedeva già un cruscotto per la visualizzazione dei dati, sul quale venivano mostrati risultati di analisi e obiettivi. Nonostante ciò, si è reso opportuno reingegnerizzarlo con nuove tecnologie che rendessero le sue funzionalità adattabili a quelle che sono al giorno d'oggi le architetture moderne con tutti i vantaggi che ne derivano, migliorando la struttura di un sistema *Legacy*<sup>1</sup> in modo tale da rendere più agevole la futura manutenibilità, aumentare le prestazioni e la velocità d'implementazione di nuove funzionalità. Per sviluppare ciò si è scelto di basarsi su un'architettura a mi-

---

<sup>1</sup>Un sistema *Legacy* è un sistema ereditato dal passato, obsoleto, che risulta difficile utilizzare nel presente ma ancora attivo.

crosservizi che comunicano usando API REST e si interfacciano tramite una Single Page Application.

### 2.1.1 Microservizi

I **microservizi** [14] sono servizi indipendenti che comunicano tra loro tramite API. È proprio l'indipendenza tra i vari servizi che costituisce un grande vantaggio nell'adottare questo approccio. Ogni servizio esegue una sola funzionalità e ciò li rende:

- **Autonomi:** possono essere sviluppati liberamente senza influenzare la struttura già esistente poiché non condividono codice con gli altri servizi;
- **Specializzati:** ogni servizio è un'implementazione ad hoc per la risoluzione di un problema specifico, permettendo anche una libertà tecnologica che porti a scegliere lo strumento migliore per ogni singola applicazione.

Inoltre i microservizi promuovono:

- **Sviluppo agile:** il software viene prodotto per incrementi rilasciando nuove versioni frequentemente;
- **Riusabilità del codice:** è possibile riutilizzare servizi per la composizione di moduli più complessi per la creazione di nuove funzionalità;
- **Resilienza:** l'errore di un servizio non preclude l'intera applicazione (come nel caso di un'architettura monolitica); il malfunzionamento può essere gestito isolando il singolo servizio senza bloccare l'intera struttura;
- **Semplicità di distribuzione:** favoriscono l'integrazione di nuovi moduli funzionali in modo semplice, potendo ripristinare il sistema alle

condizioni precedenti nel caso di errori, influenzando di meno sul costo di uno sbaglio;

- **Scalabilità:** ogni servizio è adattabile in modo indipendente in base alle proprie necessità; questo permette anche una valutazione accurata dei costi di una nuova funzionalità.

Oltre ai vantaggi già elencati, la cattiva gestione di una tale architettura può però portare anche ad una maggiore complessità nell'avere a che fare con più parti indipendenti che devono cooperare tra loro. Nonostante ciò, risulta essere la soluzione migliore rispetto al suo antagonista monolitico dove la presenza di processi dipendenti e strettamente collegati crea numerosi svantaggi, dalla gestione degli errori e quindi la modifica dell'intera applicazione alla mancanza di flessibilità per l'implementazione di nuove funzionalità.

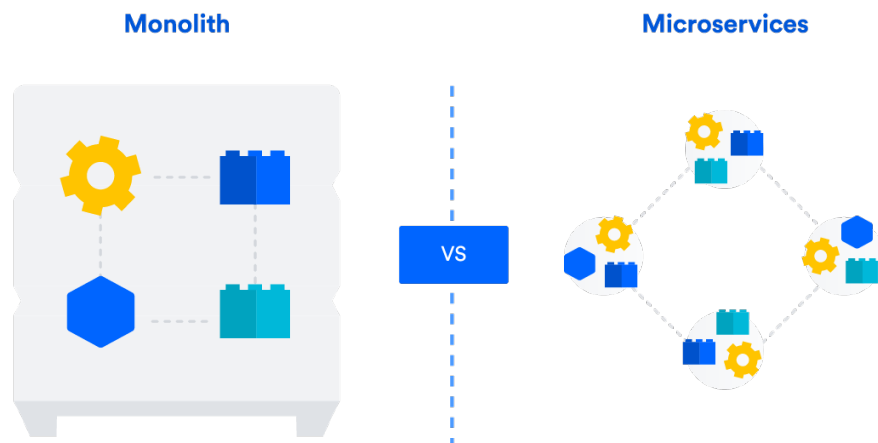


Figura 2.1: Architettura Monolitica VS Architettura a Microservizi [15]

### 2.1.2 API REST

Un'Application Programming Interface (API) definisce i protocolli di comunicazione da seguire per fare in modo che sistemi software eterogenei pos-

sano comunicare tra loro. Un'**API REST** è un'API che rispetta lo stile architetturale **REST (REpresentational State Transfer)**.

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use." (Roy Fielding. PhD thesis, 2000)

Un'architettura per essere di tipo REST deve possedere le seguenti caratteristiche [16]:

### 1. **Interfaccia Uniforme**

L'interfaccia uniforme rappresenta un formato standard di rappresentare le risorse che può differirsi dalla rappresentazione interna lato server. Ad esempio, il server può inviare i dati dal suo database in formato XML o JSON, nessuno dei quali riflette la rappresentazione interna del server. Le richieste che identificano le risorse devono utilizzare un identificatore di risorse uniforme (URI Uniform Resource Identifier) in modo tale che tutte le richieste API per la stessa risorsa siano uguali indipendentemente dall'origine della richiesta.

### 2. **Disaccoppiamento client-server**

Nella creazione di API REST il client e il server devono fungere come parti indipendenti l'una dall'altra. Il client deve essere solamente a conoscenza dell'URI per poter accedere alla risorsa desiderata. Il server invece si limita ad inviare dati tramite il protocollo HTTP.

### 3. **Stateless**

Le API REST sono stateless, ciò significa che ogni richiesta non dipende dallo stato precedente della stessa o dell'applicazione. Ogni nuova richiesta deve contenere tutte le informazioni richieste non basandosi su interazioni precedenti con il client.

### 4. **Cacheability**

Se è possibile, le risorse devono essere memorizzate in cache lato client

o in un intermediario per migliorare i tempi di risposta del server che può indicare se la risorsa fornita può essere memorizzata o meno in cache e per quanto tempo. Questa modalità alleggerisce di molto il server che non deve dare conto a richieste ridondanti.

## 5. Architettura a livelli

L'architettura a livelli permette di avere un sistema di sicurezza aggiuntivo poiché né il client né il server sanno se stanno dialogando direttamente con la parte interessata o tramite un intermediario. Ciò permette di poter utilizzare diversi server con livelli di sicurezza differenti che collaborano tra loro per poter soddisfare le richieste del client.

## 6. Codice on-demand

Le API REST possono contenere del codice eseguibile che viene trasferito al client ed eseguito localmente.

Le richieste delle API RESTful<sup>2</sup> sono formate da:

- un **identificatore della risorsa** che identifica il percorso univoco della risorsa stessa;
- un **metodo HTTP** (GET, POST, PUT, DELETE) per specificare al server che operazione deve svolgere sulla risorsa indicata;
- un **header** che contiene diverse informazioni come il formato della richiesta e il suo stato;
- possibili **parametri** e/o **dati**. Nel caso dei parametri essi forniscono al server più dettagli su ciò che deve essere fatto. I dati invece vengono utilizzati per passare informazioni quando si devono fare operazioni come POST, PUT e altri metodi HTTP.

---

<sup>2</sup>*RESTful* si riferisce a un'API che aderisce ai vincoli REST

Inoltre ogni richiesta deve essere autenticata prima di poter ricevere una risposta. L'autenticazione può avvenire in diversi modi. Il client può inviare la coppia username e password ad ogni richiesta codificandola oppure utilizzare un token (stringa di caratteri crittografata) generato dal server quando viene effettuato il login. Mettendo insieme questi due sistemi di autenticazione è possibile combinare la password e il token in modo che il sistema chieda prima la password e poi il client riceva il token identificativo per effettuare le richieste ed autenticarsi.

Le risposte invece devono contenere:

- un **codice di stato** che indichi se la richiesta è andata a buon fine o meno;
- il **corpo del messaggio** che contiene la rappresentazione della risorsa nel formato richiesto;
- un **header** contenente informazioni sul server, la codifica, la data e il tipo di documenti.

I **vantaggi** derivati dall'utilizzo di API REST sono [17]:

- **Scalabilità:**

REST ottimizza le iterazioni client-server poiché il server non deve basarsi su vecchie informazioni riguardanti le richieste dei client e il caching alleggerisce le interazioni tra le due entità favorendo la scalabilità, cioè la capacità di continuare a funzionare bene sotto un carico di lavoro crescente.

- **Flessibilità:**

I servizi REST sono flessibili grazie al disaccoppiamento client-server che permette di apportare modifiche alla piattaforma o alla tecnologia

applicata nel server non influenzando sul client. La flessibilità fa sì che sia possibile stratificare le funzioni dell'applicazione.

- **Indipendenza:**

Le API REST sono indipendenti delle tecnologie con cui sono stati sviluppati client e server, questo permette di poter modificare le tecnologie circostanti senza intaccare la comunicazione.

### 2.1.3 Single Page Application

Una **Single Page Application (SPA)** rappresenta una tipologia di progettazione delle web app che a differenza dei tradizionali siti web non è organizzato in pagine ma in una sola pagina.

In un sito tradizionale, con più pagine, ogni volta che si chiede di visualizzare una schermata il browser inoltra al server una richiesta per ricevere tutti i dati necessari che compongono la pagina, come ad esempio testi ed immagini. Tale procedura ripetuta per ogni pagina che si vuole visualizzare diventa onerosa soprattutto se i dati richiesti sono pesanti, il che implica un rallentamento nel caricamento complessivo. In Figura 2.2 è rappresentato lo schema riassuntivo del funzionamento di un sito Internet classico.

### Pagine web tradizionali



Figura 2.2: Schema riassuntivo del funzionamento di un sito Internet classico [18]

Le Single Page Application invece sono basate sull'utilizzo dei componenti, quindi non è più la pagina a cambiare interamente ma ad ogni nuova richiesta solamente i componenti o i dati che devono subire una variazione vengono ricaricati. Questo permette ad ogni nuova interazione dell'utente di non richiedere componenti comuni alle varie schermate di una web app, risparmiando flusso di dati e guadagnando in reattività. La maggiore rapidità garantisce così una esperienza di utilizzo migliore. In Figura 2.3 è rappresentato lo schema riassuntivo del funzionamento di una SPA.



### Single Page Application (SPA)

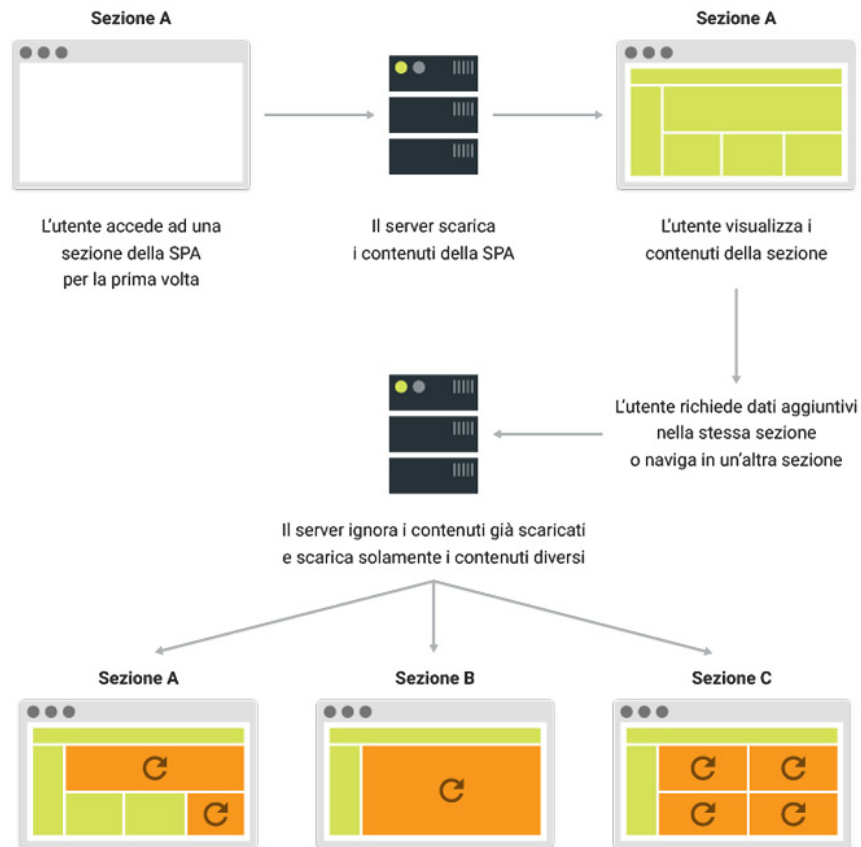


Figura 2.3: Schema riassuntivo del funzionamento di una SPA [18]

## 2.2 Dashboard di Business Intelligence

Le **dashboard di Business Intelligence** [19] favoriscono l'attuazione di processi decisionali più consapevolmente su una base di informazioni date da rappresentazioni facilmente comprensibili, quindi più utili, anche agli utenti meno esperti di analisi dei dati. Aiuta un'organizzazione a capire come capitalizzare le opportunità e affrontare i problemi.

Una dashboard di BI è importante che sia **semplice**, **intuitiva** e **funzionale**. Deve adattarsi ai vari tipi di dati da visualizzare fornendo anche la possibilità di vedere diverse rappresentazioni dello stesso dato. Implementare librerie per l'inserimento di widget, immagini e icone migliorando l'usabilità e l'impatto visivo.

Componente fondamentale per la visualizzazione dei dati sono i **grafici** che possono essere di diverso tipo (es. torta, barre, linee, ecc.) adattandosi al meglio in base all'informazione che si vuole far risaltare. I grafici possono essere affiancati da **tabelle di dati** che forniscono una panoramica dei valori dei dati rilevanti, includendo trattamenti grafici (come l'utilizzo del grassetto) per evidenziare informazioni specifiche.

Funzionalità utili sono:

- il **drill-down**, cioè la possibilità di scendere gerarchicamente facendo clic su una visualizzazione dei dati in modo da ottenere maggiori informazioni o analizzare dati più in dettaglio;
- il **filtraggio dei dati**, i filtri aiutano la ricerca e la personalizzazione dei dati da visualizzare per ottenere una visione più mirata delle informazioni presentate.

Nella fase di progettazione è opportuno identificare i dati che i futuri utilizzatori devono tenere traccia, poiché differiscono ad ogni dominio applicativo. Le possibilità di personalizzazione di una dashboard permettono di adattarla ai diversi contesti. Per questo è importante scegliere delle tecnologie di progettazione atte a sostenere una facile manutenibilità e integrazione di nuove funzionalità.

### 2.2.1 Funzionalità della Nuova Dashboard

Per il rifacimento della dashboard di Business Intelligence del prodotto Gzoom è stato scelto di riportare le stesse funzionalità che aveva precedentemente con l'aggiunta di altre personalizzazioni che ne arricchiscono l'utilizzo, mi-

gliorando l'usabilità.

### Use Case Diagram

In Figura 2.4 è possibile osservare lo use case diagram del nuovo cruscotto che mostra in maniera semplificata le varie funzionalità a cui l'utente può accedere, definite nella fase iniziale del processo di reingegnerizzazione e affinate durante lo stesso processo. L'utente identificato come 'Actor' nel diagramma, dopo aver avuto accesso al sistema ed essersi recato nel cruscotto del modulo operativo scelto, può visualizzare la lista delle varie analisi del contesto selezionato. Dopodiché cliccando su un'analisi ha la possibilità di vederne il punteggio complessivo, la lista degli obiettivi che ne fanno parte e i relativi punteggi che possono essere visualizzati sotto diverse rappresentazioni quali grafici, tachimetri o emoticon. La selezione di un obiettivo fa in modo di effettuare il drill-down, quindi scendere gerarchicamente e visualizzare sotto-obiettivi con relativi punteggi. Ad ogni obiettivo l'utente può avere la possibilità di poter consultare anche una tabella che racchiude i Key Performance Indicator e di collegarsi ad un'analisi precedente, tramite un collegamento, per poter effettuare un confronto.

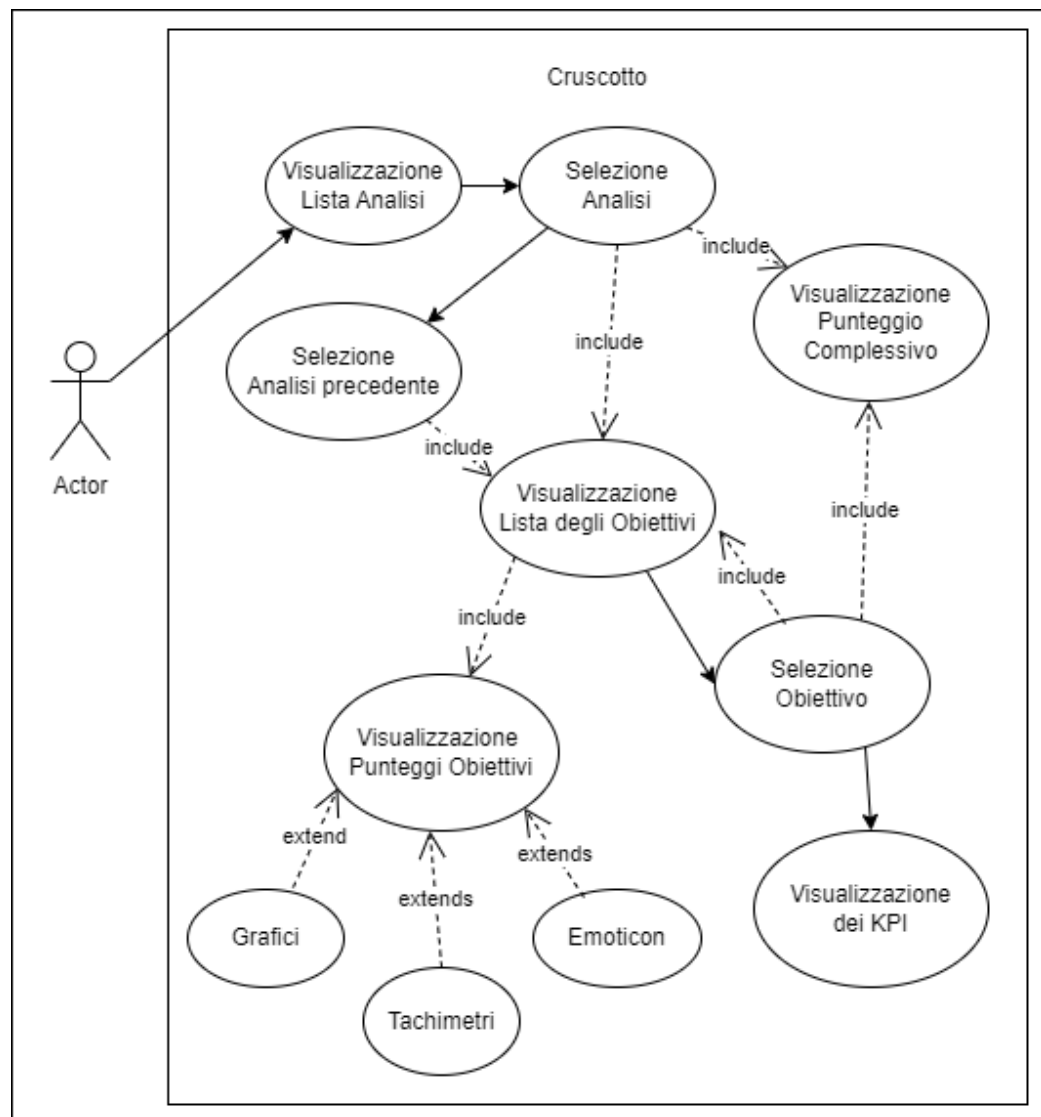


Figura 2.4: Use case diagram del Cruscotto realizzato

## Descrizione Funzionale

Le due schermate che compongono la dashboard sono la **lista delle analisi** e la **pagina degli obiettivi**. La *lista delle analisi* viene rappresentata sotto forma tabellare e racchiude tutto l'elenco delle analisi. Dato un contesto operativo e l'id dell'utente loggato il servizio ne restituisce la lista delle analisi

che l'utente è autorizzato a visualizzare.

I *contesti operativi* in Gzoom per le amministrazioni pubbliche locali si differenziano in: performance strategica, performance operativa, performance individuale, gestione processi, valutazione partecipativa, controllo di gestione, obblighi di trasparenza, privacy GDPR, rischio anticorruzione, rischio antiriciclaggio e rendicontazione sociale [20]. Ogni contesto e modulo nel sistema è identificato da un codice in modo tale da poter sapere a quale contesto e modulo si riferisce la voce di menu selezionata dall'utente, poiché ogni ambito possiede il proprio cruscotto che non differisce a livello logico ma per i dati che vengono trattati.

I *permessi* per estrarre i dati, in base alle relazioni che un utente ha con una unità organizzativa, si differenziano in questo caso in quattro livelli che indicano:

1. la visibilità limitata ai soli obiettivi legati all'unità organizzativa di cui l'utente è responsabile o delegato;
2. la visibilità limitata ai soli obiettivi legati all'unità organizzativa, ad un livello sottostante, di cui l'utente è responsabile o delegato;
3. la visibilità limitata ai soli obiettivi legati all'unità organizzativa, a due livelli sottostanti, di cui l'utente è responsabile o delegato;
4. la visibilità limitata ai soli obiettivi su cui l'utente è assegnato.

Ogni livello è identificato anch'esso da un codice che insieme all'identificativo del contesto ed altri parametri di configurazione formano una parte dei permessi di sistema che l'utente può avere.

Dalla visualizzazione delle analisi è possibile ricercarle e ordinarle attraverso dei filtri che non usano servizi, ma le informazioni vengono modificate solamente al livello di presentazione. Cliccando su un'analisi si accede alla

schermata degli obiettivi. Da questa transizione ciò che viene passato è l'identificativo dell'analisi selezionata che verrà inserito come parametro alle richieste REST per l'esecuzione dei servizi che ritornano gli obiettivi della stessa analisi.

La *pagina degli obiettivi* è formata da un'**intestazione** e da un folder contenente un possibile **elenco degli obiettivi** e un possibile **elenco degli indicatori**. Le possibili configurazioni dei componenti di questa schermata sono settate da un parametro '**comments**' che attraverso degli attributi chiave-valore ne indica la personalizzazione di come le informazioni devono essere visualizzate. Per il titolo da esporre in testata viene richiamato un servizio che cerca se è presente un obiettivo riepilogativo dell'analisi. La risposta può fornire tre risultati:

- la query non dà risultati, allora viene segnalato che non ci sono obiettivi da visualizzare;
- la query dà un solo risultato, allora diventa l'obiettivo di input di un servizio che ne ritorna il nome da esporre e degli eventuali valori di risultato;
- la query dà più risultati, allora vengono calcolati i valori di risultato come media dei risultati degli obiettivi estratti e in testata viene visualizzato l'identificativo dell'analisi.

La rappresentazione dei valori di risultato nell'intestazione è data dal parametro *mainScore* in comments che può essere stato settato con uno dei seguenti valori:

- **GAUGE**: con tale valore vengono estratti i range di colorazione (verde, giallo, rosso) e il massimale di punteggio che servono a settare la configurazione del tachimetro da visualizzare;
- **EMOTICON**: se *mainScore* ha assunto questo valore vengono valorizzate le emoticon in testata e il tipo (smile, neutral, sad) viene estratto

tramite il percorso della risorsa che viene restituito in base al valore del risultato raggiunto nel servizio di estrazione dell'obiettivo.

Il numero di valori di risultato da esporre può variare in base alla presenza o meno delle etichette relative presenti in comments che identificano il titolo che accompagna la rappresentazione del risultato. Inoltre, nella parte di testata è stata aggiunta la possibilità, tramite un button menu, di poter consultare altre analisi oltre a quella selezionata il cui id risulta specificato nei parametri *otherAnalysisId1/2/3/4* in comments.

Se tra i parametri di configurazione è valorizzato il parametro *nameDetail* allora viene attivato un folder intitolato con il valore corrispondente. Tale folder conterrà l'elenco degli obiettivi estratti tramite due possibili query: una nel caso in cui ci si trovi gerarchicamente in un primo livello, dove il drill-down è stato svolto solamente tramite l'analisi e l'altra nel caso ci si trovi ad uno step successivo scendendo verso gli obiettivi figli. In tutti e due i casi avviene un servizio di estrazione dell'elenco obiettivi ma si differenziano per parametri di input, poiché nel secondo caso contribuisce al filtraggio dei risultati, oltre all'id dell'analisi, anche l'id dell'obiettivo selezionato.

L'elenco degli obiettivi viene fornito all'utente sotto forma di tabella (contenente il nome e l'etichetta) e anche in questo caso è possibile effettuare le operazioni di filtraggio. La rappresentazione grafica dei valori di risultato segue le stesse regole definite per la testata utilizzando altri parametri ma la medesima logica per le rappresentazioni di emoticon e dei tachimetri. In questo caso il parametro che regola il comportamento di visualizzazione dei risultati è *detailScore* che oltre ad assumere i valori **EMOTICON\_LIST** e **GAUGE\_LIST**, che identificano rispettivamente la rappresentazione tramite emoticon e tachimetri, può assumere i valori **RADAR**, **PIE**, **BAR**, **LINE**, **DOUGHNUT** e **POLAR**, che rispettivamente indica una rappresentazione con l'utilizzo di un grafico dalle seguenti tipologie: radar, torta,

barre, linee, ciambella e polare. Il set dei risultati viene così utilizzato per la configurazione dei grafici che riporteranno i valori del set di dati passato. Vi è inoltre la possibilità di selezionare un tipo di grafico diverso da quello di default, che viene presettato, in modo tale che l'utente possa vedere diverse rappresentazioni dello stesso set di risultati. I grafici di tipo radar, pie, line e bar posso essere utilizzati anche in presenza di più set di dati, questo nel caso in cui i valori da valorizzare siano più di uno. Nella schermata della pagina degli obiettivi è possibile che sia abilitato un secondo folder dato dal settaggio del parametro *nameKPI* il cui valore viene inserito come nome del fascicolo che contiene l'elenco degli indicatori di risultato.

Gli *indicatori di risultato* possono essere visualizzati tramite due possibili configurazioni tabellari selezionabili tramite il parametro *detalKPI* di comments che può assumere i valori **SCORE** e **PERIOD**. In tutti e due i casi la tabella degli indicatori contiene il nome dell'indicatore e la relativa emoticon che indica la performance generale. Nel caso il parametro fosse uguale a SCORE la visualizzazione del target e del consuntivo viene sviluppato su più colonne. Se invece il parametro è uguale a PERIOD, target e consuntivo vengono visualizzati sulla stessa colonna ma divisi in due righe differenti sempre all'interno della riga dell'indicatore. Secondo altri parametri la tabella può subire delle piccole variazioni con l'aggiunta del codice dell'indicatore, della tipologia e dell'unità di misura del KPI.

Infine, cliccando su una riga della lista degli obiettivi la schermata della pagina degli obiettivi viene aggiornata mostrando gli elementi figli, i loro risultati e i dettagli dell'unità selezionata ricalcolando la loro rappresentazione sempre in base al parametro comments corrispondente.



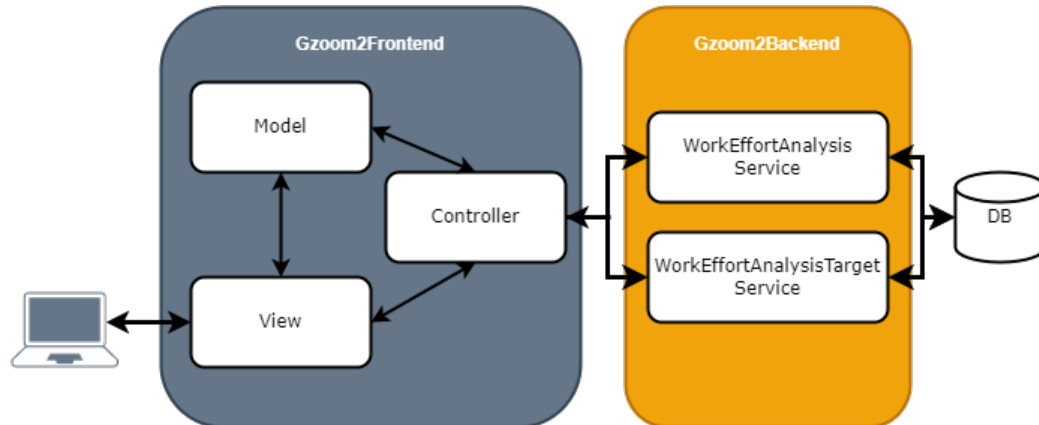
**Struttura dei servizi e gestione delle richieste/risposte**

Figura 2.5: Diagramma dell'interazione client-server

La parte di Gzoom reingegnerizzata è basata su due servizi principali uno che opera lato frontend o client e l'altro lato backend o server. Tali servizi identificati rispettivamente come **Gzoom2Frontend** e **Gzoom2Backend** interagiscono tra loro, come mostrato in Figura 2.5. Gzoom2Frontend, basato sul pattern *MVC* (*Model* - *View* - *Controller*), si occupa di fornire un'interfaccia grafica all'utente tramite la *view*, attraverso la quale il client può inoltrare richieste al server tramite l'utilizzo del *controller* che è a conoscenza solamente dell'URL identificativo della risorsa richiesta. All'interno dello stesso servizio frontend le informazioni vengono rappresentate da dei *modelli* che facilitano la manipolazione dei dati tra la *view* e il *controller*. Gzoom2Backend invece si occupa di ricevere le richieste e fornire delle risposte tramite microservizi mappati al suo interno secondo REST da un URI (Uniform Resource Identifier) e il relativo metodo HTTP che identifica l'operazione CRUD da svolgere. In Figura 2.5, all'interno del riquadro arancione, sono rappresentati i due servizi sviluppati per la dashboard di BI. Ogni servizio sviluppato all'interno di Gzoom2Backend è strutturato secondo tre livelli seguendo il pattern *Controller-Service-Repository*:

**- Controller**

Tale livello è responsabile dell'esposizione delle funzionalità che possono

essere richiamate da entità esterne; contiene tutti gli URI accessibili che identificano le risorse con i relativi metodi HTTP.

- **Service**

Il livello di servizio, accessibile solamente tramite un controller, definisce il confine dell'applicazione e il suo insieme di operazioni disponibili dal punto di vista dell'interfacciamento dei livelli client. Incapsula la logica di business dell'applicazione, controllando le transazioni e coordinando le risposte nell'implementazione delle sue operazioni; il livello di servizio interagisce con il livello di Repository per interfacciarsi con il database.

- **Repository**

Questo livello è responsabile dell'archiviazione e del recupero dati; effettua tutte le operazioni che devono essere eseguite sul database di fatti è l'unico livello che si interfaccia direttamente al DB.

In Figura 2.6 è possibile vedere l'interconnessione tra le tre parti principali che compongono un servizio, l'interazione verso al database che può avvenire solamente tramite il livello di Repository, con l'utilizzo di classi **DAO (Data Access Object)**, e l'interfacciamento delle richieste da parte del browser che possono essere introdotte attraverso il Controller. Dal browser la richiesta passa prima al Controller, poi al Service e in fine al DAO che si occupa di fare le operazioni sul database per poi fornire una risposta che viaggia fino al browser dove è partita la richiesta, passando a ritroso tra i vari componenti. Il Service che racchiude la business logic può a sua volta evocare un altro controller o chiamare più DAO per effettuare le proprie operazioni.

Per il trasferimento dati tra i vari moduli viene utilizzato un *oggetto di trasferimento dati (DTO – Data Transfer Object)* che ha il compito di creare un oggetto modello per le risposte in modo tale da facilitarne le operazioni

di elaborazione all'interno dell'ambiente prima che vengano fornite al client.

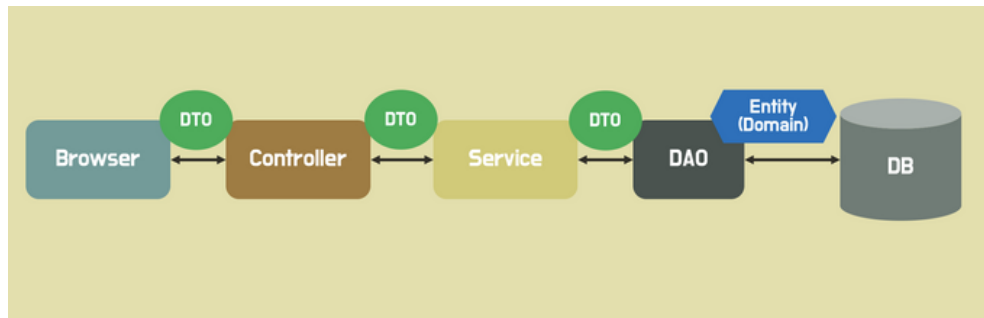


Figura 2.6: Modello Browser-Controller-Service-DAO-DB [21]

Ogni richiesta REST effettuata dal client contiene anche un token identificativo che permette al server di riconoscere l'utente loggato. Tale token è rilasciato dal server in fase di login. Il corpo di ogni risposta è invece mappato secondo il formato **JSON (JavaScript Object Notation)**. Basato su un insieme di coppie chiave-valore, questo formato è facile da leggere e scrivere per le persone ed è facile da generare e analizzare per le macchine [22].

## Capitolo 3

# Implementazione

Per la fase di realizzazione e sviluppo della dashboard, a partire dalla sua definizione logica, sono state utilizzate diverse tecnologie già definite nella fase iniziale del progetto. Scelte tecnologiche che appartengono all'insieme di soluzioni più atte a reingegnerizzare l'architettura ed a facilitarne l'implementazione con sistemi più moderni e alla completa integrazione con l'applicativo già esistente. Lo sviluppo tecnologico software permette di rimodernare il sistema con soluzioni più recenti create per facilitare il compito degli sviluppatori e aumentare le prestazioni dell'intera applicazione.

### 3.1 Backend

#### 3.1.1 Microservizi - Spring Boot

Per lo sviluppo dei microservizi è stato utilizzato il framework Spring Boot. **Spring** [23] è un framework open source leggero che utilizza il linguaggio Java. Permette di creare applicazioni di tipo enterprise<sup>1</sup> semplificando lo

---

<sup>1</sup>Un'applicazione *enterprise* (EA) è una piattaforma di sistemi software di grandi dimensioni progettata per funzionare in un ambiente come quello aziendale o governativo [24].

sviluppo e aiutando gli sviluppatori a creare applicazioni più efficaci e efficienti. **Spring Boot** è un'estensione di Spring che semplifica la costruzione di applicazioni web e microservizi tramite: autoconfigurazione, la possibilità di creare applicazioni autonome e un approccio supponente alla configurazione che tramite una serie di regole e linee guida predefinite fornisce una certa struttura e un approccio alla creazione di applicazioni, in modo tale da procurare un processo di sviluppo snello e efficiente [25].

Spring utilizza la **Dependency Injection**, cioè la capacità di un oggetto di fornire dipendenze ad un altro oggetto. In altri termini è un approccio in cui una classe utilizza funzionalità specifiche di un'altra classe. Per la configurazione dei vari oggetti, chiamati *bean*, il framework mette a disposizione delle annotazioni come in Java formattate come '@nome\_annotazione' che forniscono i metadati necessari al compilatore su come gestire una determinata classe e le sue dipendenze.

I due microservizi sviluppati *WorkEffortAnalysis* e *WorkEffortAnalysisTarget* seguono le regole di stratificazione secondo il pattern Controller-Service-Repository promuovendo la separazione delle problematiche in tre livelli. Il livello Controller si occupa di gestire le richieste in arrivo e determina quale servizio deve gestire la richiesta. Il livello Service contiene la business logic e interagisce con il livello Repository che è responsabile dell'accesso e della manipolazione dei dati memorizzati in un database. Quest'ultimo livello è caratterizzato da classi Data Access Object (DAO) che si occupano di fornire una rappresentazione orientata agli oggetti di una tabella di database e di effettuare operazioni CRUD (Create, Read, Update, Delete) per la manipolazione dei dati archiviati. Sono state create tre diverse classi che cooperano tra loro nel fornire il servizio richiesto. Le tre parti di stratificazione dei servizi sono identificabili sia dal nome che per convenzione viene composto dal nome del servizio più un suffisso -Controller, -Service o -Dao, sia dalla presenza delle annotazioni che identifica il comportamento di una determi-

nata classe nella struttura dell'applicativo.

Le principali annotazioni che sono state utilizzate per lo sviluppo dei servizi sono:

- ***@RestController***

Questo tag identifica una classe contenente le annotazioni *@RequestMapping* e *@ResponseBody*, è un'estensione dell'annotazione *@Controller* (adatta a tutti i tipi di controller, anche non HTTP). Le classi etichettate con tale annotazione identificano i controller dell'applicativo contenenti gli URL delle risorse richiamabili dal client e indica a Spring di convertire gli oggetti restituiti dai metodi della classe in formato JSON (o XML); in Figura 3.1 è riportata una parte di codice della classe *WorkEffortAnalysisController* dove è stata utilizzata l'etichetta *@RestController*;

- ***@RequestMapping***

Associa un path identificativo univoco (URL) a un metodo della classe controller o alla classe stessa; viene utilizzata per gestire le richieste HTTP in entrata associandogli il metodo del controller appropriato; tutti i metodi dei controller che sono stati sviluppati sono stati mappati per rispondere a richieste di tipo GET dato che la dashboard non prevede l'utilizzo di altri metodi HTTP poiché si limita all'estrazione dei dati e non effettua creazione, modifica e cancellazione; ad ogni *@RequestMapping* corrisponde una *@ResponseBody*; per fornire un esempio di utilizzo dell'annotazione *@RequestMapping* si può osservare in Figura 3.1 il metodo *getWorkEffortAnalysis* che viene mappato con l'URL "work-effort-analysis-id/analysisId" e metodo GET;

- ***@ResponseBody***

Specifica che il valore ottenuto dal metodo etichettato va interpretato come il contenuto della risposta; viene posto all'inizio di ogni metodo del controller come in Figura 3.1 dove l'annotazione *@ResponseBody* indica che i valori di ritorno del metodo *getWorkEffortAnalysisServi-*

ce saranno contenuti nella risposta alla richiesta avvenuta con l'URL mappato con l'annotazione *@RequestMapping* appena sopra;

- ***@PathVariable***

Nei casi in cui è stato necessario passare dei parametri per ricevere una risposta personalizzata è stata utilizzata questa annotazione che specificando il nome dell'attributo variabile ne estrae il valore assunto e lo assegna alla variabile denominata successivamente; in Figura 3.1 è possibile osservare un esempio di come l'annotazione *@PathVariable* sia stata usata nel metodo *getWorkEffortAnalysis* specificando l'attributo variabile, in questo caso *'analysisId'*, identificabile anche nel path identificativo del metodo tra parentesi graffe, al cui è stato attribuito la variabile *'analysisId'* di tipo *String*;

```
@RestController
@RequestMapping(value = "", produces = { MediaType.APPLICATION_JSON_VALUE })
public class WorkEffortAnalysisController {

    private final WorkEffortAnalysisService workEffortAnalysisService;

    @Autowired
    public WorkEffortAnalysisController(WorkEffortAnalysisService workEffortAnalysisService) {
        this.workEffortAnalysisService = workEffortAnalysisService;
    }

    @RequestMapping(value = "work-effort-analysis-id/{analysisId}", method = RequestMethod.GET)
    @ResponseBody
    public WorkEffortAnalysis getWorkEffortAnalysis(@PathVariable(value = "analysisId") String analysisId) {
        ...
    }
}
```

Figura 3.1: Parte di codice sorgente della classe *WorkEffortAnalysisController*

- ***@Service***

Identifica le classi che compongono la business logic dell'applicativo che viene separata dal *@RestController*; l'annotazione *@Service* viene anche utilizzata per le classi DAO se anch'esse compiono operazioni di logica aziendale;

- ***@Autowired***

È l'annotazione che permette di utilizzare la *Dependency Injection*

che può essere utilizzata nei costruttori, nei campi e nei metodi permettendo di creare un'istanza delle classi dipendenti, indicando quali sono le dipendenze richieste da un determinato oggetto; in Figura 3.2 viene riportata una parte di codice della classe *WorkEffortAnalysisService* dove l'etichetta *@Autowired* viene utilizzata nel costruttore della classe in modo tale da indicare la dipendenza con la classe *WorkEffortAnalysisDao*, la quale viene richiamata per le operazioni di CRUD;

```
@Service
public class WorkEffortAnalysisService {
    private static final Logger LOG = getLogger(WorkEffortAnalysisService.class);

    private final WorkEffortAnalysisDao workEffortAnalysisDao;

    @Autowired
    public WorkEffortAnalysisService(WorkEffortAnalysisDao workEffortAnalysisDao) {
        this.workEffortAnalysisDao = workEffortAnalysisDao;
    }

    public WorkEffortAnalysis getWorkEffortAnalysis(String workEffortAnalysisId) {
        return workEffortAnalysisDao.getWorkEffortAnalysis(workEffortAnalysisId);
    }
}
```

Figura 3.2: Parte di codice sorgente della classe *WorkEffortAnalysisService*

- ***@Transactional***

Annotazione utilizzata per i metodi DAO per definire l'ambito di una singola transazione di database; in Figura 3.3 è raffigurato un esempio di utilizzo dell'annotazione *@Transactional* dove è applicata al metodo *getWorkEffortAnalysisWithContext* che esegue la query di estrazione della lista delle analisi dato un contesto operativo.



```
@Service
public class WorkEffortAnalysisDao {

    private static final Logger LOG = getLogger(WorkEffortAnalysisDao.class);

    private final SQLQueryFactory queryFactory;

    private final PermissionService permissionService;

    @Autowired
    public WorkEffortAnalysisDao(SQLQueryFactory queryFactory, PermissionService permissionService) {
        this.queryFactory = queryFactory;
        this.permissionService = permissionService;
    }

    /**
     * This function gets the work effort analyses with a context.
     *
     * @param context Context.
     * @param userLoginId User login id.
     * @return List of work effort analysis.
     */
    @Transactional
    public List<WorkEffortAnalysis> getWorkEffortAnalysesWithContext(String context, String userLoginId) {
        ...
    }
}
```

Figura 3.3: Parte di codice sorgente della classe WorkEffortAnalysisDao

### 3.1.2 Query

#### QueryDSL

L'applicativo Gzoom si può interfacciare con diversi motori di database (ad esempio MySQL, PostgreSQL, ecc.) grazie anche all'utilizzo di QueryDSL.

**QueryDSL** [26] è un framework Java che consente la generazione di query indipendenti dai tipi, utilizzando una sintassi simile a SQL. Permette la portabilità poiché le query non dipendono dal database e dal particolare dialetto SQL che assume.

Per la costruzione delle query QueryDSL fa capo ai tipi cosiddetti Q-Type che fanno direttamente riferimento alle classi che identificano le entità presenti nell'applicativo (classi con l'annotazione *@Entity*), ma sono precedute dalla lettera Q. Viene istanziata un'istanza SQLQueryFactory a cui successivamente viene concatenata la query di estrazione con le relative clausole

anch'esse concatenate l'un all'altra come è possibile notare nella porzione di codice in Figura 3.4. Dopo di che avviene la conversione, o *bindings*, e l'esecuzione della query al cui risultato viene associata una classe, o un'aggregazione delle stesse, DTO che identifica i campi che ritorna l'interrogazione appena eseguita. Infine, il DTO viene trasferito al metodo chiamante al livello di servizio.

```
QWorkEffortAnalysis qWA = QWorkEffortAnalysis.workEffortAnalysis;
QWorkEffortType qWAType = QWorkEffortType.workEffortType;

QUserLoginSecurityGroup qULSG = QUserLoginSecurityGroup.userLoginSecurityGroup;
QSecurityGroupPermission qSGP = QSecurityGroupPermission.securityGroupPermission;

String permission = ContextPermissionPrefixEnum.getPermissionPrefix(context);

SQLQuery<WorkEffortAnalysis> tupleSQLQuery = queryFactory.select(qWA).from(qWA)
    .innerJoin(qWAType).on(qWA.workEffortTypeId.eq(qWAType.workEffortTypeId))
    .where ((qWAType.parentTypeId.eq(context))
        .and ((queryFactory.selectOne().from(qULSG)
            .innerJoin(qSGP).on((qULSG.groupId.eq(qSGP.groupId))
                .and(qSGP.permissionId.eq(permission + "MGR_ADMIN"))))
            .where(qULSG.userLoginId.eq(userLoginId)
                .and(qWA.availabilityId.eq("AVAIL_INTERNAL"))))
            .exists().or(qWA.availabilityId.eq("AVAIL_PUBLIC"))));

SQLBindings bindings = tupleSQLQuery.getSQL();
LOG.info("{} ", bindings.getSQL());
LOG.info("{} ", bindings.getNullFriendlyBindings());
QBean<WorkEffortAnalysis> wa = bean(WorkEffortAnalysis.class, qWA.all());
List<WorkEffortAnalysis> ret = tupleSQLQuery.transform(GroupBy.groupBy(qWA.workEffortAnalysisId).list(wa));
```

Figura 3.4: Query di estrazione della lista delle analisi dato un contesto

## MyBatis

Per la costruzione di query più complesse è stato utilizzato il framework MyBatis. **MyBatis** [27] è basato su un file XML detto Mapper al cui interno si possono definire i tag per l'inserimento di query SQL, che vengono lasciate al programmatore, e di come deve essere mappata la risposta dell'interrogazione in un determinato oggetto.

All'interno del file denominato *workEffortAnalysisTargetMapper.xml* sono state definite alcune delle query richiamabili dal servizio *WorkEffortAnalysisTarget* che fa capo alla schermata della lista degli obiettivi. Per fare ciò sono stati utilizzati i seguenti tag:

- **<resultMap>**: descrive come caricare gli oggetti dai set di risultati del database, contiene un id e un tipo che si riferisce alla classe DTO con la quale deve essere mappata la risposta; all'interno del tag *<resultMap>* sono poi definiti tramite i tag *<result>* e *<id>*; in Figura 3.5 è riportato un esempio di utilizzo;
- **<result>**: tali tag definiscono i nomi delle colonne che riporta la risposta alla query, il tipo corrispondente nel database e la proprietà della classe DTO a cui deve corrispondere quella determinata colonna;
- **<id>**: questo tag a differenza del tag *<result>* mappa la colonna come proprietà identificatrice da utilizzare quando si confrontano le istanze.

```
<resultMap id="BaseResultMap" type="it.mapsgroup.gzoom.mybatis.dto.WorkEffortAnalysisTarget">
  <id column="work_effort_id" jdbcType="VARCHAR" property="workEffortId"/>
  <result column="etch" jdbcType="VARCHAR" property="workEffortEtch"/>
  <result column="work_effort_name" jdbcType="VARCHAR" property="workEffortName"/>
  <result column="work_effort_name_lang" jdbcType="VARCHAR" property="workEffortNameLang"/>

  <result column="sc_amount" jdbcType="DECIMAL" property="scAmount"/>
  <result column="rvc_icon_content_id" jdbcType="VARCHAR" property="rvcIconContentId"/>
  <result column="drc_object_info" jdbcType="VARCHAR" property="drcObjectInfo"/>

  <result column="st_amount" jdbcType="DECIMAL" property="stAmount"/>
  <result column="rvt_icon_content_id" jdbcType="VARCHAR" property="rvtIconContentId"/>
  <result column="drt_object_info" jdbcType="VARCHAR" property="drtObjectInfo"/>

  <result column="sc2_amount" jdbcType="DECIMAL" property="sc2Amount"/>
  <result column="rvc2_icon_content_id" jdbcType="VARCHAR" property="rvc2IconContentId"/>
  <result column="drc2_object_info" jdbcType="VARCHAR" property="drc2ObjectInfo"/>

  <result column="st2_amount" jdbcType="DECIMAL" property="st2Amount"/>
  <result column="rvt2_icon_content_id" jdbcType="VARCHAR" property="rvt2IconContentId"/>
  <result column="drt2_object_info" jdbcType="VARCHAR" property="drt2ObjectInfo"/>
</resultMap>
```

Figura 3.5: tag resultMap workEffortAnalysisTargetMapper.xml

- **<sql>**: identifica un blocco riutilizzabile di SQL a cui possono fare riferimento altre istruzioni tramite un attributo id richiamabile tramite

il tag `<include>` che, come attributo, specificherà l'id di riferimento al blocco sql corrispondente; in Figura 3.6 è presente una parte di query racchiusa dal tag `<sql>`, che viene spesso utilizzata nelle interrogazioni al database presenti sul file *workEffortAnalysisTargetMapper.xml*;

```
<sql id="aliasAmountContentObject">
    SC.AMOUNT AS SC_AMOUNT, RVC.ICON_CONTENT_ID AS RVC_ICON_CONTENT_ID, DRC.OBJECT_INFO AS DRC_OBJECT_INFO,
    ST.AMOUNT AS ST_AMOUNT, RVT.ICON_CONTENT_ID AS RVT_ICON_CONTENT_ID, DRT.OBJECT_INFO AS DRT_OBJECT_INFO,
    SC2.AMOUNT AS SC2_AMOUNT, RVC2.ICON_CONTENT_ID AS RVC2_ICON_CONTENT_ID, DRC2.OBJECT_INFO AS DRC2_OBJECT_INFO,
    ST2.AMOUNT AS ST2_AMOUNT, RVT2.ICON_CONTENT_ID AS RVT2_ICON_CONTENT_ID, DRT2.OBJECT_INFO AS DRT2_OBJECT_INFO
</sql>
```

Figura 3.6: tag sql workEffortAnalysisTargetMapper.xml

- `<select>`: contiene un'interrogazione SQL di estrazione dati; oltre ad un attributo id, che deve corrispondere al nome del metodo chiamante, contiene l'attributo resultMap dove va specificato l'id della mappa con il quale deve essere mappata la risposta e parameterType che identifica il tipo dei parametri che vengono passati alla query; la Figura 3.7 mostra un utilizzo del tag `<select>`, dove l'id corrisponde al metodo chiamante *getWorkEffortAnalysisTargetHeaderOne* e i risultati vengono mappati tramite il tag resultMap con id '*BaseResultMap*'.

```
<select id="getWorkEffortAnalysisTargetHeaderOne" resultMap="BaseResultMap" parameterType="java.util.Map">
    SELECT WE.WORK Effort_ID, WE.WORK Effort_NAME, WE.WORK Effort_NAME_LANG,
    <include refid="aliasAmountContentObject"/>
    FROM WORK Effort WE
    INNER JOIN WORK Effort_Type WET ON WET.WORK Effort_Type_ID = WE.WORK Effort_Type_ID
    ...
    WHERE WA.WORK Effort_Analysis_ID = #{analysisId} -- INPUT.WORK Effort_Analysis_ID
    AND WE.WORK Effort_ID = #{workEffortId} -- INPUT.WORK Effort_ID
</select>
```

Figura 3.7: tag select workEffortAnalysisTargetMapper.xml

Le query in questo caso non sono più eseguite direttamente dalla classe identificabile come DAO, ma sono richiamate da quest'ultima tramite un'interfaccia che riceve i parametri da passare alla query e li identifica con una stringa che se richiamata (`{nome_stringa}`) all'interno della query viene rimpiazzata con il valore corrispondente. In questo caso l'interfaccia funge da mezzo di comunicazione tra il DAO e il file XML fornendo dei metodi richiamabili i cui nomi corrispondono all'identificativo della query da eseguire.

## 3.2 Frontend

### 3.2.1 Angular

L'intero servizio Gzoom2Frontend è basato su Angular. **Angular** [28] è un framework per la creazione di Single Page Application lato client che utilizza i linguaggi HTML e TypeScript. Oltre alle funzionalità di base può implementare librerie TypeScript esterne.

Gli elementi costruttivi principali di un'applicazione Angular sono i **componenti**. Ogni componente è definito, come in Figura 3.8, da una classe implementata in un file TypeScript, che contiene i dati e la logica, a cui è associata un modello HTML che ne definisce la vista.

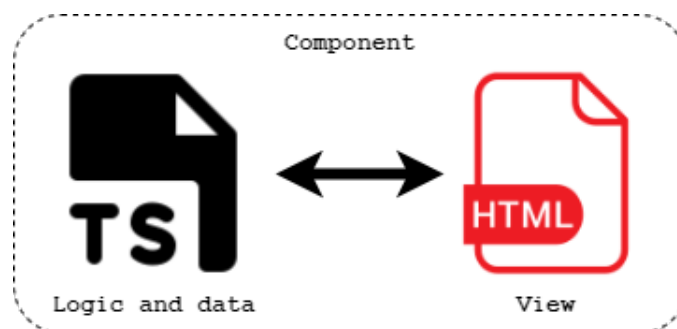


Figura 3.8: File principali che compongono un componente

I componenti Angular sono definiti con il decoratore **@Component**. I decoratori sono posizionati prima della definizione di una classe e caratterizzati dal prefisso '@'. Un decoratore fornisce i metadati alla classe, ai suoi metodi ed alle sue proprietà. I metadati forniscono maggiori informazioni su come il framework deve elaborare la classe. Altri decoratori utilizzati nello sviluppo sono stati:

- **@NgModule**: identifica una classe in cui si dichiarano i moduli, servizi e unità funzionali utilizzati da un determinato componente;
- **@Injectable**: contrassegna una classe come disponibile per essere fornita e inserita come dipendenza;
- **@Input**: se associato a una proprietà consente a un componente padre di passare i dati a un componente figlio;
- **@Output**: viceversa, consente al figlio di inviare dati a un componente padre.

Le due schermate del cruscotto realizzato sono un'aggregazione di singoli componenti indipendenti a cui vengono passati dati da visualizzare o dati utili per la loro configurazione. Tali componenti sono a loro volta racchiusi all'interno di un altro componente e così via, come un effetto matrioska, fino ad arrivare al componente radice che racchiude tutta l'applicazione.

Ogni componente possiede un **ciclo di vita** che inizia quando viene creata l'istanza della classe corrispondente, continua con il rilevamento delle modifiche e termina quando Angular distrugge l'istanza. Per poterne regolare il comportamento in ogni stato del ciclo di vita vengono utilizzati dei metodi appositi. In particolare sono stati implementati i seguenti metodi:

- **ngOnInit()**: metodo richiamato nella fase di inizializzazione del componente ed eseguito una sola volta, quindi utile per la fase di configurazione iniziale;
- **ngOnChanges()**: metodo richiamato quando viene impostata o reimpostata una proprietà di input e implementando un argomento *SimpleChanges* può essere usato per rilevare la differenza tra il vecchio e il nuovo valore.

Nel componente contenitore delle due schermate realizzate, dopo la sua generazione, è stato configurato il **routing** che permette di passare da una schermata all'altra tramite l'utilizzo di un path identificativo che renderizza al componente corrispondente. In una Single Page Application si modifica ciò che l'utente vede mostrando o nascondendo i vari componenti. Ciò significa che man mano che gli utenti interagiscono con l'applicazione si spostano in diverse visualizzazioni già definite. Per passare da una vista all'altra, Angular utilizza il modulo *Router* che usa gli URL come istruzione per cambiare visualizzazione.

Avere diverse personalizzazioni del cruscotto è stato possibile grazie alle **direttive strutturali** messe a disposizione dal framework, per poter mostrare o meno parti di codice o componenti che dipendono dai parametri di configurazione della dashboard tramite l'utilizzo di variabili, rendendo l'applicazione dinamica. In questo modo, ad esempio, se si predilige la visualizzazione dei grafici per la visualizzazione dei risultati degli obiettivi la view viene renderizzata escludendo le altre configurazioni. Esempi di direttive strutturali sono *ngIf*, *ngSwitch* e *ngFor*. Inoltre, tramite all'**Angular Template Syntax** il framework effettua il *binding*, cioè l'accoppiamento dei dati alla view in modo tale da poter specificare nei tag html attributi con valori stringa definiti nella classe del componente.

Su Gzoom2Frontend ad ogni componente principale che identifica una scher-

mata è poi associato un *service*, il quale gestisce le chiamate REST e la traduzione delle risposte JSON mappandole in modelli che rispecchiano i campi ricevuti, in modo tale da tipizzare gli oggetti ricevuti per una maggiore semplicità di manipolazione dei dati. Ogni classe service può essere richiamata per l'utilizzo dei suoi metodi che tramite un servizio chiamato *ApiClientService* inoltra la richiesta di una risorsa al server, specificando il metodo HTTP, il path identificativo e i relativi parametri se presenti. In Figura 3.9 è riportato un esempio di un metodo della classe service del cruscotto realizzato. Il metodo `getWorkEffortAnalysisWithContext()`, ricevuto come parametro il contesto operativo, manda una richiesta GET con l'URL specifico per richiamare il servizio di estrazione della lista delle analisi dato un contesto. La risposta JSON viene poi mappata in un array di tipo `WorkEffortAnalysis`, che rappresenta la model, e poi restituito al metodo chiamante sotto forma di array `Observable`.

```
/**
 * Gets the list of analyses in a context.
 *
 * @param context - Context.
 * @returns Observable array of WorkEffortAnalysis.
 */
getWorkEffortAnalysisWithContext(context: string): Observable<WorkEffortAnalysis[]> {
  console.log('search workEffortAnalysis list');
  return this.client
    .get(`work-effort-analysis/${context}`).pipe(
      map(json => json.results as WorkEffortAnalysis[])
    );
}
```

Figura 3.9: Metodo service `getWorkEffortAnalysisWithContext()`

L'**observer** pattern è un modello di progettazione software in cui un oggetto 'soggetto' (subject) mantiene un elenco di 'osservatori' (observers) e notifica a loro ogni cambiamento di stato. È un pattern simile al publish/subscribe, ma non identico. Gli Observables favoriscono il passaggio dei messaggi tra le varie parti che compongono l'applicazione, gestiscono al meglio gli eventi, la programmazione asincrona e la gestione di più valori.



Per poter invece avere un'interfaccia più uniforme e agevolare lo sviluppo si è scelto di accostare ad Angular il framework **PrimeNG** [29] che contiene una raccolta di componenti, per la User Interface, pronti per l'utilizzo. I componenti di primeNG sono utilizzabili dopo aver installato la libreria, tramite comando CLI `'npm install primeng --save'`, e aver importato i moduli corrispondenti nella propria applicazione. Svolti questi passaggi vengono utilizzati i tag corrispondenti e come dei semplici componenti.

### 3.2.2 Tachimetri - Ngx-gauge

Per l'implementazione dei tachimetri è stata utilizzata una libreria esterna ad Angular chiamata **ngx-gauge** [30]. Essa fornisce un componente di misurazione personalizzabile per app e dashboard. Dopo l'installazione del modulo corrispondente tramite istruzione CLI `'npm install --save ngx-gauge'` e la sua importazione specificando tale modulo nel file che identifica le dipendenze del componente che lo utilizza, è stato possibile configurarlo tramite dei parametri che ne curano l'aspetto grafico, come il tipo, le dimensioni e le colorazioni, ed altri parametri che settano il valore attuale dell'indicatore ed il valore massimo e minimo raggiungibile.

Il componente è stato configurato in modo tale che possa ricevere come parametro un'interfaccia appositamente creata che ne permette di definire il valore dell'etichetta riferita al valore del tachimetro, il valore che deve assumere l'indicatore, il suo valore massimo, il suo valore minimo e i range per le colorazioni di rosso, giallo e verde che indicano se il valore ottenuto da un obiettivo è da considerarsi soddisfacente o meno. In Figura 3.10 è possibile osservare l'interfaccia sviluppata.

```
export interface gaugeConfig {  
  gaugeLabel?: string,  
  gaugeValue?: number,  
  gaugeMax?: number,  
  gaugeMin?: number,  
  fromValueRed?: number,  
  fromValueYellow?: number,  
  fromValueGreen?: number,  
}
```

Figura 3.10: Interfaccia Tachimetro

La classe che gestisce i parametri del tachimetro a sua volta implementa un metodo `ngOnChanges` che viene richiamato ogni qual volta i parametri di input variano in modo tale da aggiornare i valori di configurazione che vengono passati al componente installato settandone la view. In Figura 3.11 è raffigurato il passaggio dei parametri per il settaggio della view del tachimetro.

```
<ngx-gauge  
  aria-label="chart"  
  [size]="gaugeSize"  
  [type]="gaugeType"  
  [thick]="gaugeThick"  
  [value]="gaugeValue"  
  [min]="gaugeMin"  
  [max]="gaugeMax"  
  [cap]="gaugeCap"  
  [label]="gaugeLabel"  
  [append]="gaugeAppendText"  
  [markers]="markerConfig"  
  [thresholds]="thresholdConfig"  
  [foregroundColor]="foregroundColor"  
  [margin]="gaugeMargin"  
  [duration]="gaugeDuration"></ngx-gauge>
```

Figura 3.11: Passaggio parametri al modulo ngx-gauge

I parametri presenti in Figura 3.11 ma non presenti nell'interfaccia in Figura

3.10 sono invece configurazioni statiche che ne definiscono la grandezza, il tipo (cerchio, semi-cerchio, arco), gli aspetti grafici e di animazione. In Figura 3.12 sono mostrati i settaggi iniziali del tachimetro.

```

gaugeSize = 115;           /*Specifies the size of the canvas in which Gauge will be drawn.
                             It is used as width and height both.*/
gaugeType = "arch";        //Specifies the gauge's type.
gaugeThick = 10;           //Specifies the thickness of the gauge's bar.
gaugeValue = 10;           /*Specifies the current value of the Gauge in the range specified by
                             min and max. It is a required attribute.*/
gaugeMin = 0;              //Specifies the minimum numeric value for gauge's scale.
gaugeMax = 100;            //Specifies the maximum numeric value for gauge's scale.
gaugeDuration = 500;       //Specifies the duration (in milliseconds) of the Gauge's animation.
gaugeCap = "butt";         //The style of line ending at the gauge's end.
gaugeLabel = "";           //Specifies the text to display below the Gauge's reading.
gaugeAppendText = "";      /*Specifies a string appended to the Gauge's reading.
                             For example "%" most commonly used.*/
gaugeMargin = 20;          //Specifies an optional margin for the gauge.
sizeMarkerLine = 2;        //Marker line size
sizeMarkerLineNum = 4;     //Marker line size with label
foregroundColor="green"    //Specifies the foreground color of the Gauge's scale.

```

Figura 3.12: Variabili di configurazione del tachimetro

In Figura 3.13 è possibile vedere un esempio grafico del tachimetro realizzato.



Figura 3.13: UI Tachimetro

### 3.2.3 Grafici - Chart.js

**Chart.js** [31] è la libreria JavaScript open source utilizzata per la creazione di diversi tipi di grafici interattivi e animati creati tramite l'elemento canvas di HTML5. Chart.js permette di visualizzare dei dataset tramite l'utilizzo di grafici a linee, a barre, radar, polari, ad area, a torta e ad anello, dando anche

la possibilità di personalizzarli e modificarli a piacimento [32]. La libreria fornisce un costruttore che accetta diversi parametri di configurazione, tra i più importati ci sono i parametri `type`, `data` e `options`.

- **type**: definisce la tipologia di grafico da creare, citandone alcuni: `bar`, `pie`, `line`;
- **data**: contiene i dataset da rappresentare e le etichette corrispondenti, quest'ultime contenute in un array di stringhe;
- **options**: può racchiudere tutte le personalizzazioni del grafico come il titolo, la posizione della legenda, la configurazione degli assi, le animazioni, ecc.

I grafici utilizzati per la visualizzazione dei risultati che fanno riferimento a degli obiettivi sono stati realizzati raccogliendo i vari valori da mostrare in set di dati e poi passati alla classe `ChartComponet` con l'utilizzo di una interfaccia (Figura 3.14) che oltre ai set di dati dà la possibilità di inserire le etichette corrispondenti da visualizzare e di specificare la tipologia di grafico.

```
export interface chartConfig {  
  type?: string,  
  labels?: string[],  
  etch?: string[],  
  rangeMaxName?: string,  
  dataMax?: number[],  
  dataCon?: number[],  
  dataTar?: number[],  
  dataSC?: number[],  
  dataST?: number[],  
  dataSC2?: number[],  
  dataST2?: number []  
  scoreEtch1?: string,  
  scoreEtch2?: string,  
  scoreEtch3?: string,  
  scoreEtch4?: string,  
}
```

Figura 3.14: Interfaccia Grafici

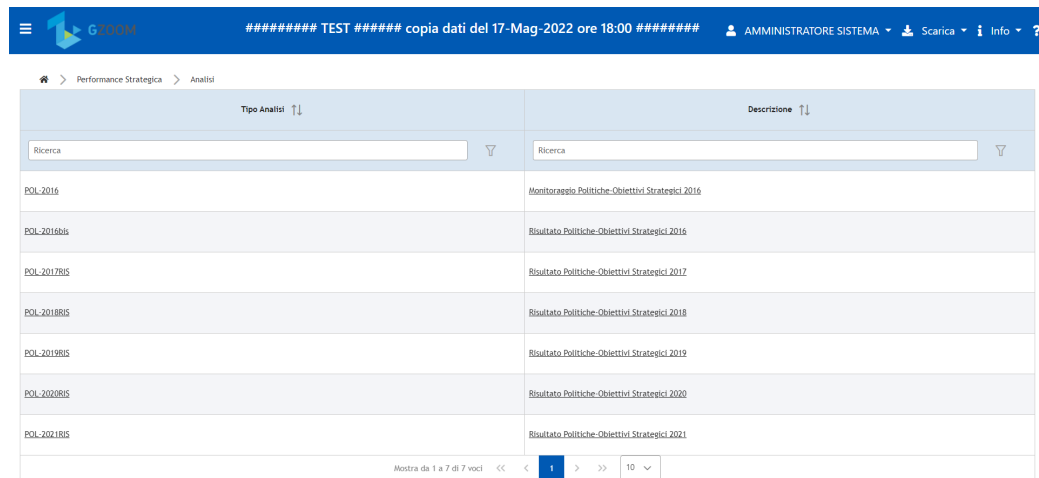
I dati ricevuti in input vengono poi elaborati per costruire i parametri da passare al modulo di chart.js per l'elaborazione e la costruzione del grafico indicato.

## Capitolo 4

### Risultati

I risultati ottenuti tramite la reingegnerizzazione della dashboard di Business Intelligence sono visibili principalmente tramite interfaccia grafica, componente fondamentale per la rappresentazione di informazioni che aiutino enti e organizzazioni nel processo decisionale per il raggiungimento dei propri obiettivi.

In Figura 4.1 è possibile osservare un esempio della schermata della lista delle analisi realizzata, consultabile dopo che l'utente abbia selezionato la voce di menu corrispondente, all'interno del modulo e del contesto operativo scelto. La lista delle analisi risulta essere rappresentata sotto forma tabellare. Le informazioni che ne vengono mostrate sono il tipo dell'analisi e la corrispondente descrizione. L'utente in questa schermata può effettuare una ricerca tra le analisi filtrando o ordinando alfabeticamente la lista tramite le funzioni di filtraggio presenti appena sotto l'intestazione della tabella.



Tipo Analisi ↑↓	Descrizione ↑↓
<input type="text" value="Ricerca"/>	<input type="text" value="Ricerca"/>
POL_2016	Monitoraggio Politiche-Obiettivi Strategici 2016
POL_2016bis	Risultato Politiche-Obiettivi Strategici 2016
POL_2017bis	Risultato Politiche-Obiettivi Strategici 2017
POL_2018bis	Risultato Politiche-Obiettivi Strategici 2018
POL_2019bis	Risultato Politiche-Obiettivi Strategici 2019
POL_2020bis	Risultato Politiche-Obiettivi Strategici 2020
POL_2021bis	Risultato Politiche-Obiettivi Strategici 2021

Mostra da 1 a 7 di 7 voci << 1 >> 10

Figura 4.1: Schermata della lista delle analisi

Un'altra azione consentita è la possibilità di cliccare su una determinata analisi per poter scendere nel dettaglio e visualizzare quelli che sono gli obiettivi che la compongono, i risultati da essi raggiunti e il risultato complessivo dell'analisi. A tal punto si passa alla schermata della lista degli obiettivi, che può essere visualizzata secondo diverse configurazioni.

La schermata della lista degli obiettivi è composta da un'intestazione e da un folder. L'intestazione, rappresentata nella fascia superiore della schermata, contiene le informazioni riguardanti l'ultima analisi o l'ultimo obiettivo cliccato (nel caso di drill-down nella lista degli obiettivi), specificando il nome dell'obiettivo o il tipo di analisi e i suoi relativi valori di completamento che possono venire mostrati utilizzando un tachimetro (come in Figura 4.2) o un emoticon card (come in Figura 4.3 o 4.4).

Il folder può possedere due sezioni differenti, una che mostri la lista degli obiettivi con la relativa rappresentazione grafica dei risultati valorizzati, e una contenente gli indicatori di performance KPI. La sezione con la lista degli obiettivi può essere visualizzata, in base ai parametri di configurazione, sia con l'ausilio di un grafico (Figura 4.2) sia con l'utilizzo di tachimetri (Fi-

gura 4.3) o emoticon card (Figura 4.4), che mostrano lo stato di avanzamento dei singoli obiettivi. In tutti i casi è presente una tabella contenente l'elenco degli obiettivi dove è possibile fare operazioni di filtraggio e ordinamento come in ogni tabella implementata nella dashboard. Cliccando su un obiettivo della tabella vengono aggiornati i dati dei vari componenti della schermata, con le informazioni riguardanti l'obiettivo cliccato e i suoi obiettivi figli.

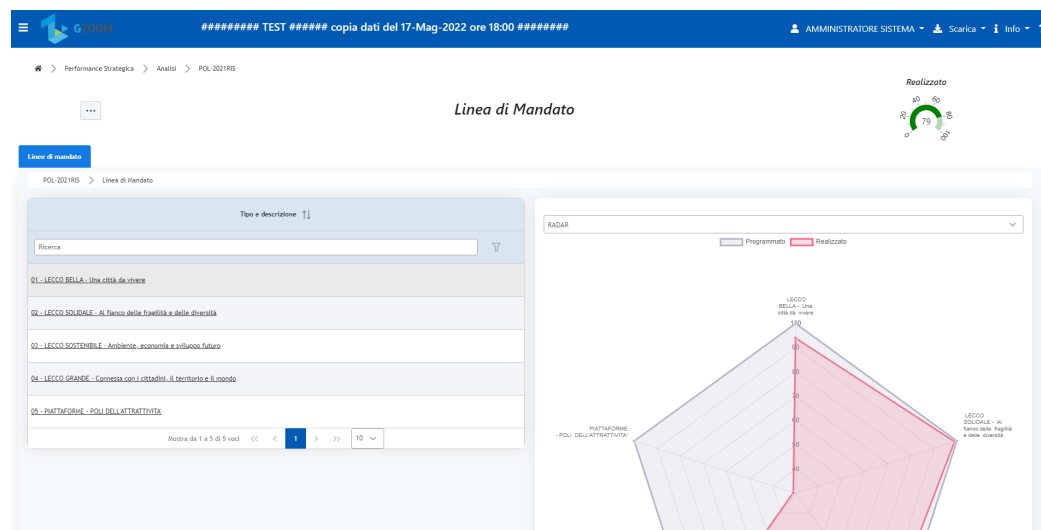


Figura 4.2: Schermata della lista degli obiettivi con grafico

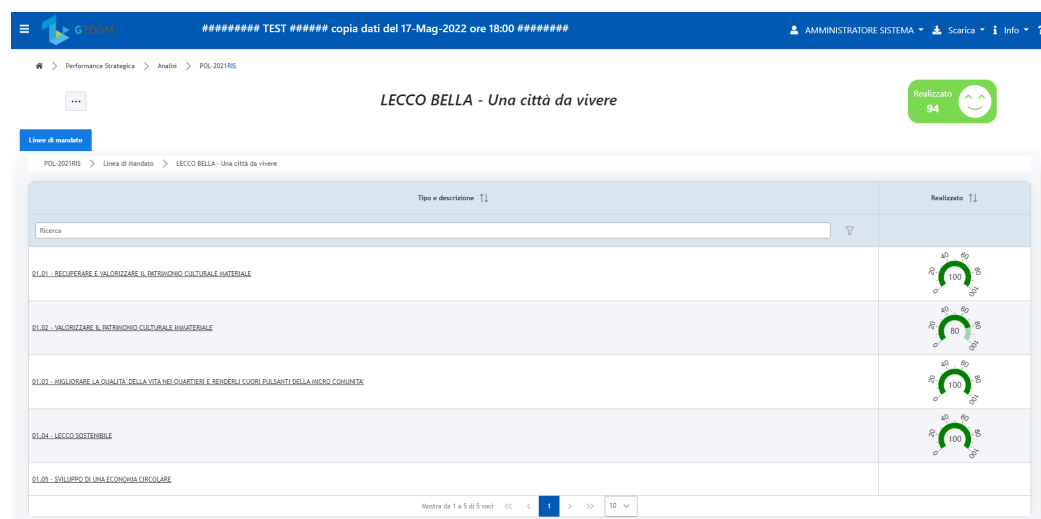


Figura 4.3: Schermata della lista degli obiettivi con tabella di tachimetri





Figura 4.4: Schermata della lista degli obiettivi con tabella di emoticon card

In caso in cui la rappresentazione dei risultati avvenga tramite grafici, l'utente ha la libertà di scegliere tramite un menù a tendina, come in Figura 4.5 , con che tipo di grafico visualizzare lo stesso set di dati.

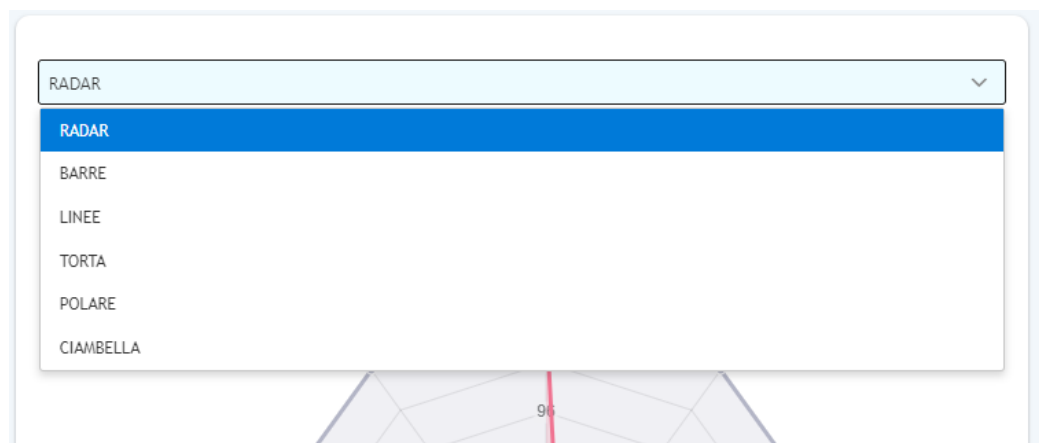


Figura 4.5: Menù di selezione della tipologia di grafico

I tipi di grafici selezionabili e visualizzabili sono: a radar (Figura 4.6), a barre (Figura 4.7), a linee (Figura 4.8), a torta (Figura 4.9), polare (Figura 4.10) e a ciambella (Figura 4.11).

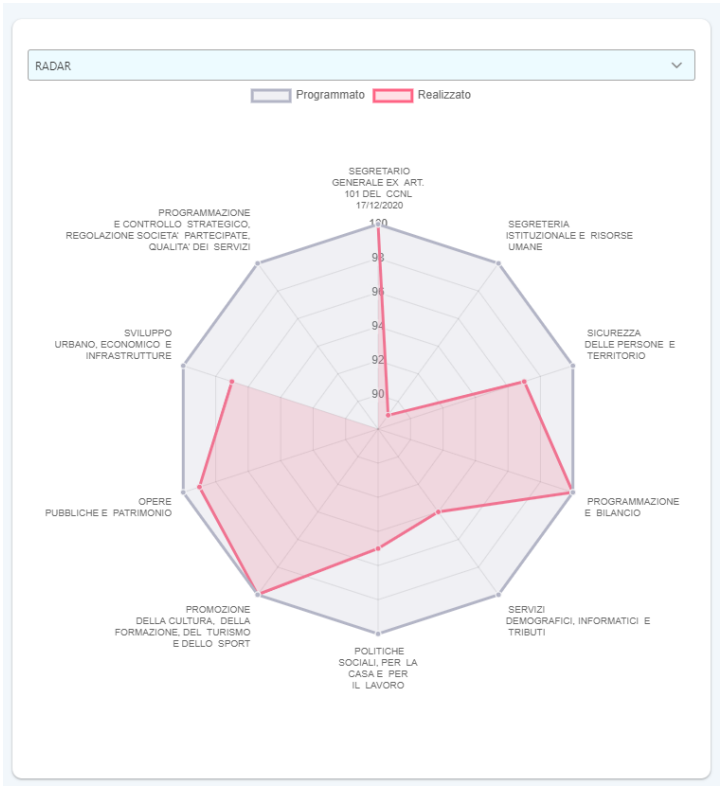


Figura 4.6: Grafico a radar

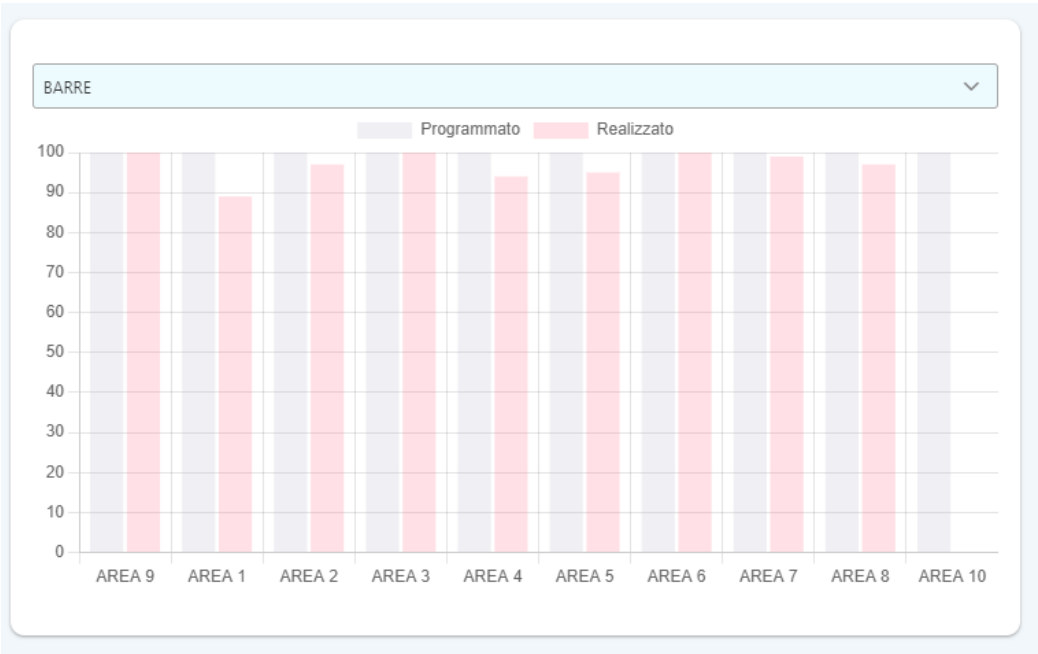


Figura 4.7: Grafico a barre



Figura 4.8: Grafico a linee

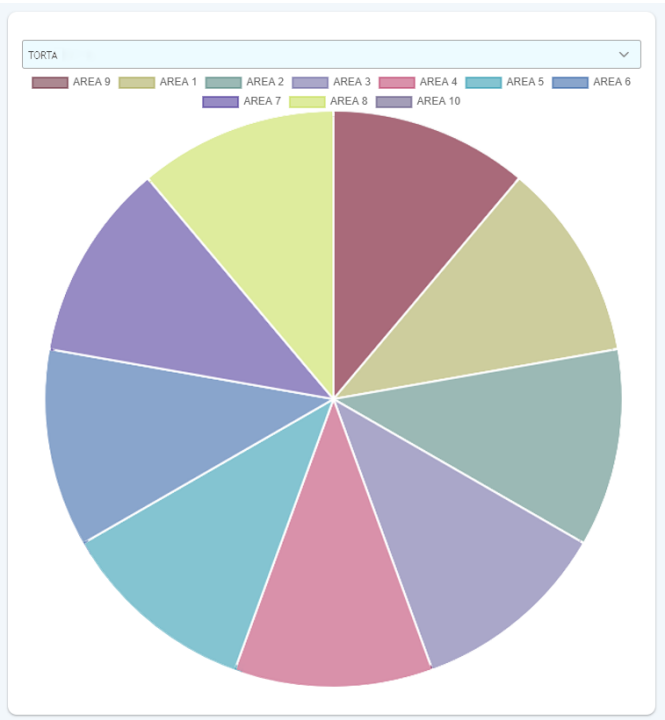


Figura 4.9: Grafico a torta

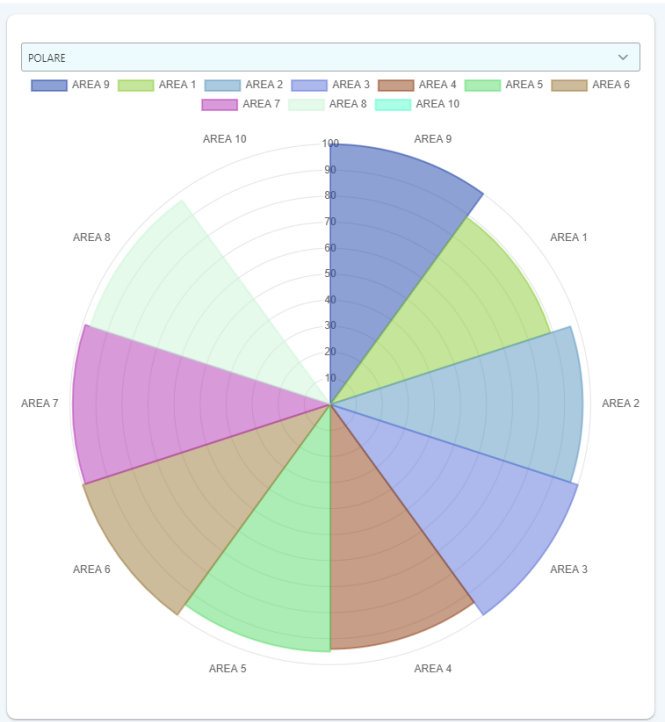


Figura 4.10: Grafico polare

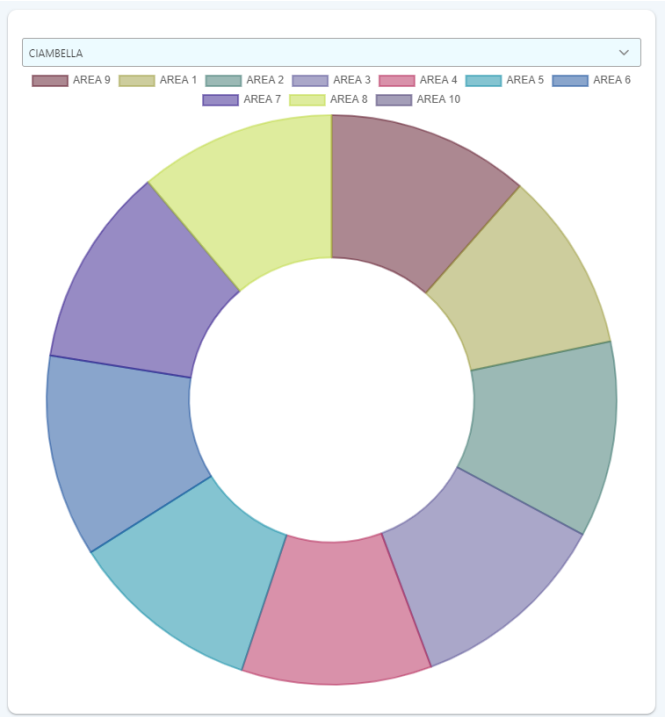


Figura 4.11: Grafico a ciambella

Invece, l'utilizzo di tachimetri o emoticon card permette, oltre a valorizzare i risultati, di avere un giudizio generale sullo stato attuale dell'avanzamento di un obiettivo. Per fare ciò viene assimilato un colore tra rosso, arancione e verde, che indichi se il valore ottenuto rappresenta un buon risultato (colore verde), quindi vicino al completamento dell'obiettivo, oppure un valore medio (colore arancione) o basso (colore rosso) nel caso in cui si debba ancora lavorare su quel determinato obiettivo per ottenere un buon risultato. Nel caso dell'emoticon card ai corrispettivi tre colori corrisponde una faccina che può essere felice (colore verde), neutrale (colore arancione) o triste (colore rosso). Per i tachimetri, diversamente, vengono riportate le colorazioni in trasparenza in range variabili che permettono di osservare quanto un risultato sia più o meno vicino all'inizio o alla fine di una determinata colorazione. Ad esempio, in Figura 4.12, dato al tachimetro un valore di 60, l'indicatore assume il colore arancione poiché si trova nel range dei valori medi, ma come si nota dalla trasparenza dei colori il valore è molto vicino all'ottenimento di un buon risultato data la vicinanza alla fascia verde.



Figura 4.12: Tachimetro

La sezione degli indicatori di performance KPI contiene invece una tabella con la lista degli indicatori con il loro relativo stato di avanzamento facendo capo e riportando i valori di target e consuntivo di ogni indicatore, includendo anche una rappresentazione dei risultati ottenuti tramite emoticon card, come in Figura 4.13. A tale tabella possono essere aggiunte, tramite configurazione,

colonne in cui si espone l'unità di misura e la tipologia dell'indicatore, o aggiungere un codice dello stesso come prefisso prima del nome del KPI.



Figura 4.13: Schermata dei Key Performance Indicator

Nell'intestazione della schermata della lista degli obiettivi è anche presente un menù identificato dall'icona dei tre puntini che fornisce un collegamento diretto alla consultazione di altre analisi, come ad esempio quella dell'anno precedente.

A livello di reattività nel caricamento della dashboard è stato notato un notevole miglioramento rispetto al cruscotto precedente. Lo sviluppo tramite componenti indipendenti, con una struttura di una Single Page Application (SPA), ha permesso al browser client di renderizzare l'interfaccia grafica in modo asincrono, eseguendo più attività contemporaneamente senza bloccarsi. In questo modo l'applicazione può gestire più richieste contemporaneamente, senza attendere il completamento di una richiesta prima di elaborarne un'altra, fornendo un'esperienza rapida e reattiva all'utente.

L'utilizzo di diversi framework ha permesso di utilizzare la tecnologia più adatta per ogni compito. Inoltre, la rimodernizzazione dell'architettura ha fatto sì che la dashboard di BI possa implementare e fornire nuove funzionalità con un tempo di time-to-market ridotto.

## Conclusioni e sviluppi futuri

L'utilizzo dei dati è fondamentale affinché le organizzazioni abbiano successo e rimangano un passo avanti rispetto alla concorrenza. Sfruttando i dati, le organizzazioni possono migliorare le proprie operazioni, favorire la crescita e l'innovazione. L'utilizzo di dashboard di Business Intelligence (BI) risulta fondamentale in questo ambito per migliorare il processo decisionale, permettendo alle organizzazioni di identificare tendenze, modelli e relazioni che possono migliorare le strategie che guidano a una costante crescita. I dati possono aiutare le organizzazioni a semplificare i processi e a ridurre gli sprechi in termini di risorse.

La reingegnerizzazione della dashboard di BI di Gzoom ha permesso di modernizzare una componente fondamentale dell'applicativo che aiuta a tenere sotto controllo gli obiettivi che le organizzazioni si impongono e i risultati che ottengono. Questo processo di cambiamento è stato regolato dalla necessità di migliorare la struttura di un sistema Legacy che presenta numerosi svantaggi per l'implementazione di nuove funzionalità, poiché basato su un sistema monolitico. Lo sviluppo della nuova dashboard basata su un'architettura a microservizi e l'utilizzo di diverse tecnologie e pattern di sviluppo che puntano a creare indipendenza tra le parti dell'applicativo ha aiutato a creare un cruscotto personalizzabile, reattivo, graficamente più accattivante e soprattutto in linea con quelle che sono le architetture moderne garantendo una più agevole manutenibilità futura e una maggiore velocità d'implementazione di nuove funzionalità.

Lo sviluppo e l'evoluzione delle dashboard di Business Intelligence (BI) vengono regolati:

- dalle nuove tecnologie, che cambiano il modo di progettare, sviluppare e utilizzare una dashboard di BI fornendo nuove capacità o funzionalità;
- dalle mutevoli esigenze aziendali, che variano in continuazione inizialmente per un processo di digitalizzazione per poi passare alla raffinazione e rimodernizzazione di sistemi già esistenti, poiché i sistemi obsoleti non trovano più posto nell'era moderna dove fanno da padroni velocità, portabilità e interoperabilità; ma anche per un solo fattore di conformità a quelli che sono i cambiamenti normativi imposti alle varie organizzazioni o enti;
- dall'evoluzione delle preferenze degli utenti, rappresentato da un cambiamento di esigenze, aspettative e desideri, dato anche dal continuo sviluppo della tecnologia; gli utenti si aspettano dashboard di BI che siano interattive, visivamente accattivanti e accessibili da più dispositivi; a supporto del processo decisionale, gli utenti possono richiedere la possibilità di permettere a team di lavorare insieme e prendere decisioni basate sugli stessi dati o avere a disposizione analisi più avanzate, come ad esempio l'analisi predittiva o l'analisi aumentata.

La dashboard di Business Intelligence realizzata si occupa di mostrare lo stato attuale di un determinato obiettivo che un ente si è posto, dando un quadro generale su quelli che sono i valori raggiunti in quel determinato istante di consultazione. L'implementazione di algoritmi basati sull'analisi predittiva permetterebbe alle organizzazioni di fare previsioni sulle tendenze e sui risultati futuri sulla base di dati storici. Inoltre, tenendo traccia dell'andamento di un obiettivo nel tempo sarebbe possibile mostrare una sua rappresentazione che arricchirebbe le informazioni che l'utente consulta per un processo decisionale più consapevole.



L'accessibilità di una dashboard di BI, cioè la possibilità di accedervi indipendentemente da dove si trova l'utente o dal dispositivo utilizzato, ad oggi risulta un tema fondamentale poiché gli utenti con la crescita dei dispositivi mobili interagiscono sempre di più con dispositivi come smartphone e tablet. Il cruscotto realizzato è stato sviluppato per l'utilizzo attraverso display di PC desktop, laptop o schermi con dimensioni simili, ma non risulta ottimizzato per i dispositivi con display ridotto. Creando un layout dinamico di tipo responsive, che in base alle dimensioni e all'orientamento del dispositivo su cui viene visualizzato si adatta, utilizzando una grafica scalabile dove le dimensioni dei caratteri e altri elementi visivi, come ad esempio immagini, tabelle e grafici, vengono elaborati, ridimensionati e riorganizzati in modo tale da rendere il tutto più leggibile, permetterebbe agli utenti piena libertà di poter consultare le informazioni con l'utilizzo del dispositivo che più gli è comodo in quel determinato momento.

# Bibliografia

- [1] Maps Group. Cosa facciamo. <https://mapsgroup.it/azienda/cosa-facciamo/>.
- [2] Alessandro Rezzani. Le 3 v dei big data: Caratteristiche. <https://www.dataskills.it/le-tre-v-dei-big-data/>, Gen 2018.
- [3] Silvia Sanna. Big data e pubblica amministrazione: Perché sono importanti? <https://contrattipubblici.org/blog/2019/08/22/big-data-pubblica-amministrazione-perche-importanti/#:~:text=I%20Big%20Data%20sono%20fondamentali,dei%20processi%20decisionali%20dell'Ente.,> Ago 2019.
- [4] Fabrizio Caldiroli. Che cos'è la business intelligence e come può aiutare un'azienda. <https://universeit.blog/business-intelligence/>, Gen 2022.
- [5] John Daintith. *IT*. Oxford University Press, 2009.
- [6] Eleonora Truzzi. Business intelligence: Cos'è e come può migliorare la tua azienda. <https://www.nextre.it/business-intelligence/>, Nov 2021.
- [7] SDD\_Gzoom. Software design document. <https://artexe.atlassian.net/wiki/spaces/GzoomTecnici/pages/2263711925/SDD+Gzoom+Standard>.

- [8] Ignazio Ballai. Modello organizzativo 231: Cos'è e a cosa serve. <https://studiolegaleballai.it/2021/03/16/modello-organizzativo-231-cosa-serve/>, Apr 2021.
- [9] Gzoom. Che cos'è. <https://artexe.mapsgroup.it/data-driven-governance/gzoom/>, Mar 2020.
- [10] Apache ofbiz. <https://ofbiz.apache.org/>.
- [11] Randy Stafford. Service layer. <https://martinfowler.com/eaCatalog/serviceLayer.html>.
- [12] Spring boot architecture - javatpoint. [https://www.javatpoint.com/spring-boot-architecture#:~:text=Persistence%20Layer%3A%20The%20persistence%20layer,%2C%20delete\)%20operations%20are%20performed](https://www.javatpoint.com/spring-boot-architecture#:~:text=Persistence%20Layer%3A%20The%20persistence%20layer,%2C%20delete)%20operations%20are%20performed).
- [13] ckan.org. Ckan - the open source data management system. <https://ckan.org/>.
- [14] Joseph R. Roach. Cosa sono i microservizi? <https://aws.amazon.com/it/microservices/#:~:text=I%20microservizi%20sono%20un%20approccio,controllati%20da%20piccoli%20team%20autonomi.,> 2007.
- [15] Chandler Harris. Microservizi e architettura monolitica a confronto. <https://www.atlassian.com/it/microservices/microservices-architecture/microservices-vs-monolith>.
- [16] IBM Cloud Education. Rest-apis. <https://www.ibm.com/it-it/cloud/learn/rest-apis>, Apr 2021.
- [17] Joseph R. Roach. Cos'è un'api restful? <https://aws.amazon.com/it/what-is/restful-api/>, 2007.

- [18] Niccolò Maria Menozzi. Che cos'è una single page application (spa). <https://www.dreamonkey.com/it/blog/che-cos-e-una-single-page-application-spa/>, Nov 2021.
- [19] Shaun Sutner. What is a business intelligence dashboard (bi dashboard)? <https://www.techtarget.com/searchbusinessanalytics/definition/business-intelligence-dashboard>, Ago 2020.
- [20] Gzoom Value Governance. Amministrazioni pubbliche locali – gzoom value governance:. <https://gzoom.mapsgroup.it/amministrazioni-pubbliche-locali/>.
- [21] hudi.blog. Dto. <https://hudi.blog/data-transfer-object/>.
- [22] JSON. Introduzione a json. <https://www.json.org/json-it.html>.
- [23] Spring. <https://spring.io/>.
- [24] Icy Science. Che cos'è un'applicazione enterprise? <https://it.theastrologypage.com/enterprise-application>, Gen 1970.
- [25] IBM. What is java spring boot? <https://www.ibm.com/topics/java-spring-boot>.
- [26] Timo Westkämper. Querydsl unified queries for java. <http://querydsl.com/>.
- [27] Clinton Begin. Mybatis. <https://mybatis.org/mybatis-3/>.
- [28] Angular. <https://angular.io/>.
- [29] Primeng. <https://primeng.org/>.
- [30] NGX gauge a customizable gauge component for angular 4 and beyond. ngx-gauge. <https://ashish-chopra.github.io/ngx-gauge/>.
- [31] Chart.js. <https://www.chartjs.org/>.

- 
- [32] GeekandJob. Cos'è chart.js e a cosa serve. <https://www.geekandjob.com/wiki/chart-js>.

