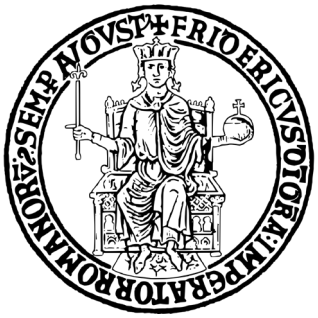
UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II



SCUOLA POLITECNICA E DELLE TECNICHE DI BASE

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di laurea in Informatica

L'importanza del modello del CMMI e la realizzazione di un CMMI Estimator

Relatore:

Prof. Porfirio Tramontana

Candidato:

Giannantonio Buonocore

Matricola: N86/1978

Anno Accademico 2021-2022

INDICE

Introduzione

Il seguente lavoro di tesi nasce da uno studio di quasi tre mesi, in collaborazione con NTT Data Italia.NTT DATA è una multinazionale con sede a Tokyo che si occupa di system integration, servizi professionali e consulenza strategica facente parte del gruppo Nippon Telegraph and Telephone (NTT).Serve principalmente i mercati telecomunicazioni, servizi, multi-utility, finanziari, pubblico, manifatturiero e della sanità.

Il motivo del mio studio nasce dall' importanza che sta avendo e che sempre più avrà il processo CMMI nel mondo lavorativo indipendentemente dalla grandezza della realtà aziendale. Non a caso risulta essere utilizzato in progetti di qualunque entità ed è popolare in molti settori. Noti sono i contributi che sta dando nel ridurre i rischi nello sviluppo di prodotti, nei servizi e in particolar modo nel software engineering.

Il mio studio si basa su come sviluppare un CMMI Estimator.

Inizialmente verrà spiegato nel dettaglio cos'è il CMMI, come è nato e che sviluppi ha avuto nel tempo.

Successivamente verrà introdotto il perché è stato sviluppato un CMMI Estimator, qual è l'idea per la sua realizzazione e quali sono stati i processi di raccolta dei requisiti che mi sono serviti per la realizzazione dell'estimatore.

Nell'intermezzo verranno esposte le soluzioni e le considerazioni tecnologiche e le funzionalità che vanno a comporre questo strumento. Si arriverà alla fine dove verranno menzionate le tempistiche per lo studio e la realizzazione del progetto.

L'elaborato si pone l'obiettivo di spiegare la realizzazione di un CMMI Estimator ed evidenziare il suo rilievo nello sviluppo di un prodotto.

1. CMMI

1.1 Cos'è il CMMI

Il CMMI (Capability Maturity Model Integration) è un processo e un modello comportamentale che aiuta le organizzazioni a ottimizzare il miglioramento dei processi e incoraggiare comportamenti produttivi ed efficienti che riducono i rischi nello sviluppo di software, prodotti e servizi. Il CMMI è stato sviluppato dal Software Engineering Institute della Carnegie Mellon University come strumento di miglioramento dei processi per progetti, divisioni o organizzazioni.

È progettato per aiutare a migliorare le prestazioni fornendo alle aziende tutto ciò di cui hanno bisogno per sviluppare costantemente prodotti e servizi migliori. Ma il CMMI è più di un modello di processo; è anche un modello comportamentale. Le aziende possono infatti utilizzare il CMMI anche per affrontare la logistica del miglioramento delle prestazioni sviluppando benchmark misurabili e può anche aiutare a creare una struttura per incoraggiare comportamenti produttivi ed efficienti in tutta l'organizzazione.

Il CMMI è stato sviluppato per combinare più modelli di maturità aziendale in un unico framework. Il CMMI è un framework metodologico integrato di modelli CMM sviluppato tra il 1987 e il 1997. CMMI Versione 1.1 è stato rilasciato nel 2002, seguito dalla Versione 1.2 nel 2006 e dalla Versione 1.3 nel 2010; la V1.3 è stata sostituita dalla V2.0 a marzo 2018.

Nella sua prima iterazione come Software il modello è stato adattato all'ingegneria del software. Le versioni successive del CMMI sono diventate più astratte e generalizzate, consentendone l'applicazione allo sviluppo di hardware, software e servizi in ogni settore. Con il rilascio della V2.0, il processo è stato semplificato.

In precedenza, il CMMI si occupava di tre aree di interesse, tra cui:

1. Lo sviluppo di prodotti e servizi;
2. La creazione di servizi;
3. L' acquisizione di prodotti e servizi.

Tutte queste aree sono state fuse in un modello autonomo.

Ogni iterazione mira a essere più facile da comprendere e utilizzare per le aziende rispetto alla precedente e ogni modello è progettato per essere più conveniente e più facile da integrare o distribuire, incoraggiando le aziende a concentrarsi sulla qualità rispetto alla quantità stabilendo parametri di riferimento per il controllo di venditori e fornitori, identificando e risolvendo i problemi di processo, riducendo al minimo i rischi e costruendo una cultura aziendale che supporterà il modello CMMI.

1.2 Livelli di maturità

I livelli di maturità di CMMI sono:

- Livello di maturità 0, incompleto: in questa fase il lavoro può o non può essere completato. Gli obiettivi non sono stati fissati e i processi sono formati solo in parte o non soddisfano le esigenze organizzative;
- Livello di maturità 1, iniziale: i processi sono visti come imprevedibili e reattivi. In questa fase, il lavoro viene completato ma spesso è in ritardo e fuori budget. Questa è la fase peggiore in cui un'azienda può trovarsi: un ambiente imprevedibile che aumenta il rischio e l'inefficienza;
- Livello di maturità 2, gestito: è stato raggiunto un livello di gestione del progetto. I progetti sono pianificati, eseguiti, misurati e controllati a questo livello, ma ci sono ancora molte questioni da affrontare;
- Livello di maturità 3, definito: in questa fase, le organizzazioni sono più proattive che reattive. Esiste una serie di standard a livello di organizzazione per fornire una guida tra progetti, programmi e portfolio. Le aziende comprendono le loro carenze, come affrontarle e qual è l'obiettivo del miglioramento;
- Livello di maturità 4, gestione quantitativa: questa fase è più misurata e controllata. L'organizzazione sta elaborando dati quantitativi per determinare processi prevedibili in linea con le esigenze degli stakeholder. L'azienda è in anticipo sui rischi, con maggiori informazioni basate sui dati sulle carenze dei processi;

- Livello di maturità 5, ottimizzazione: qui i processi di un'organizzazione sono stabili e flessibili. In questa fase finale, un'organizzazione sarà in costante stato di miglioramento e risposta ai cambiamenti o ad altre opportunità. L'organizzazione è stabile, il che consente maggiore agilità e innovazione in un ambiente prevedibile.

1.3 Livelli di capacità

Il CMMI ha anche livelli di capacità che vengono utilizzati per valutare le prestazioni di un'organizzazione e il miglioramento dei processi, in quanto si applica a un'area pratica individuale delineata nel modello CMMI e può aiutare a strutturare il processo e a migliorare le prestazioni.

I livelli di capacità sono:

- Livello di capacità 0, incompleto: prestazioni incoerenti e un approccio incompleto per soddisfare l'intento dell'area pratica.
- Livello di capacità 1, iniziale: la fase in cui le organizzazioni iniziano ad affrontare i problemi relativi alle prestazioni in un'area pratica specifica, ma in cui non esiste ancora un insieme completo di pratiche in atto.
- Livello di capacità 2, gestito: i progressi stanno iniziando a manifestarsi e c'è una serie completa di pratiche in atto che riguardano specificamente il miglioramento nell'area pratica.
- Livello di capacità 3, definito: c'è un focus sul raggiungimento degli obiettivi di progetto e prestazioni organizzative e ci sono chiari standard organizzativi in atto per affrontare i progetti in quell'area pratica.

L'ultima versione del CMMI (la 2.0) si concentra maggiormente sull'impatto delle prestazioni sul business e su come comprendere le esigenze di prestazioni di un'organizzazione. Sono disponibili informazioni su come stabilire obiettivi di prestazione e quindi tenere traccia di tali obiettivi per assicurarsi che vengano raggiunti a tutti i livelli di maturità aziendale.

Il CMMI V2.0 mira anche a ridurre il costo complessivo delle valutazioni e ad abbreviare il tempo necessario per la valutazione e l'organizzazione. Il CMMI V2.0 riduce inoltre la quantità di conoscenze tecniche incluse e risulta quindi più comprensibile per chi non fa parte del settore tecnologico. C'è anche una piattaforma online in cui gli utenti possono costruire e progettare un modello che si adatti alle esigenze specifiche dell'organizzazione.

Il CMMI Institute ha incluso ulteriori informazioni su come dimostrare il ROI (return on investment, rapporto tra il risultato operativo e il capitale investito netto), in modo che i leader possano coinvolgere altri dirigenti. I benchmark e gli obiettivi delle prestazioni delineati nel CMMI possono aiutare le aziende a garantire che tutti i progetti e i processi siano convenienti o redditizi.

1.4 Categorie di strumenti

Nel mondo CMMI esistono molteplici strumenti. Il tipo di strumento CMMI che funzionerà meglio per una generica organizzazione dipenderà dalle esigenze dell'azienda. Seguendo il CMMI, si identificano gli strumenti migliori per le nostre esigenze durante il Livello di maturità 2 o 3.

Le categorie di strumenti possono includere:

- Gestione di progetti e documenti;
- Bug tracker;
- Stima;
- Gestione dei requisiti e del design;
- Strumenti di decisione e analisi;
- Strumenti di metrica;
- Applicazione di integrazione.

Il mercato degli strumenti del mondo CMMI è ben fornito e vario. Si va da software che sono adattati a questi tipi di valutazione, come ad esempio Excel, a strumenti professionali specifici per queste tematiche.

Tra questi va citato, essendo anche il più quotato, il software Excellence Suite della SoftExpert, un software che aiuta le aziende a seguire l'approccio CMMI, riducendo i costi di conformità, massimizzando il successo, aumentando la produttività e riducendo i rischi.

Le soluzioni di SoftExpert consentono alle organizzazioni di seguire facilmente CMMI, fornendo risorse per gestire processi, audit, non conformità, KPI di qualità, record di qualità e report, aumentando l'efficienza organizzativa, riducendo le rilavorazioni e gli sprechi.

2. Raccolta dei requisiti

La raccolta dei requisiti è stata possibile attraverso vari incontri avvenuti con esponenti di NTT Data.

È stato prefissato che il progetto del CMMI Estimator avrebbe dovuto seguire fedelmente lo schema di un documento che mi è stato consegnato: quest'ultimo contiene varie informazioni in formato tabellare.
Su ogni tabella devono essere possibili diverse operazioni:

- Inserimento;
- Aggiornamento;
- Eliminazione;
- Ricerca.

Inoltre, si deve implementare un meccanismo per il calcolo delle stime seguendo un percorso a parte nell'applicazione rispetto alle funzionalità precedentemente citate.
Su mia iniziativa, ho pianificato un'ulteriore funzionalità che permettesse al singolo utente la stampa della stima a lui interessata.

Preso atto di questo, analizzando il documento e comprendendo le necessità aziendali ho optato, per quanto riguarda la tipologia del software, per la realizzazione di una web app. La scelta delle tecnologie per la sua realizzazione è stata fatta rispettando l'andamento e le richieste del mercato odierno affinché l'applicazione sia al passo coi tempi e non risulti obsoleta.

Per la modalità di protezione dei dati la mia scelta è ricaduta sul realizzare nell'application un login, così da permettere l'uso dell'estimator solo a chi autorizzato o per mantenere traccia dei vari accessi.
Si è deciso che per l'accesso all'applicazione sarebbe stato consono utilizzare credenziali aziendali, nello specifico l'e-mail e la password personale dei dipendenti.

Per l'interfaccia grafica, ho attuato un approccio user friendly, per permettere agli utenti un rapido apprendimento e un rapido utilizzo dello strumento.

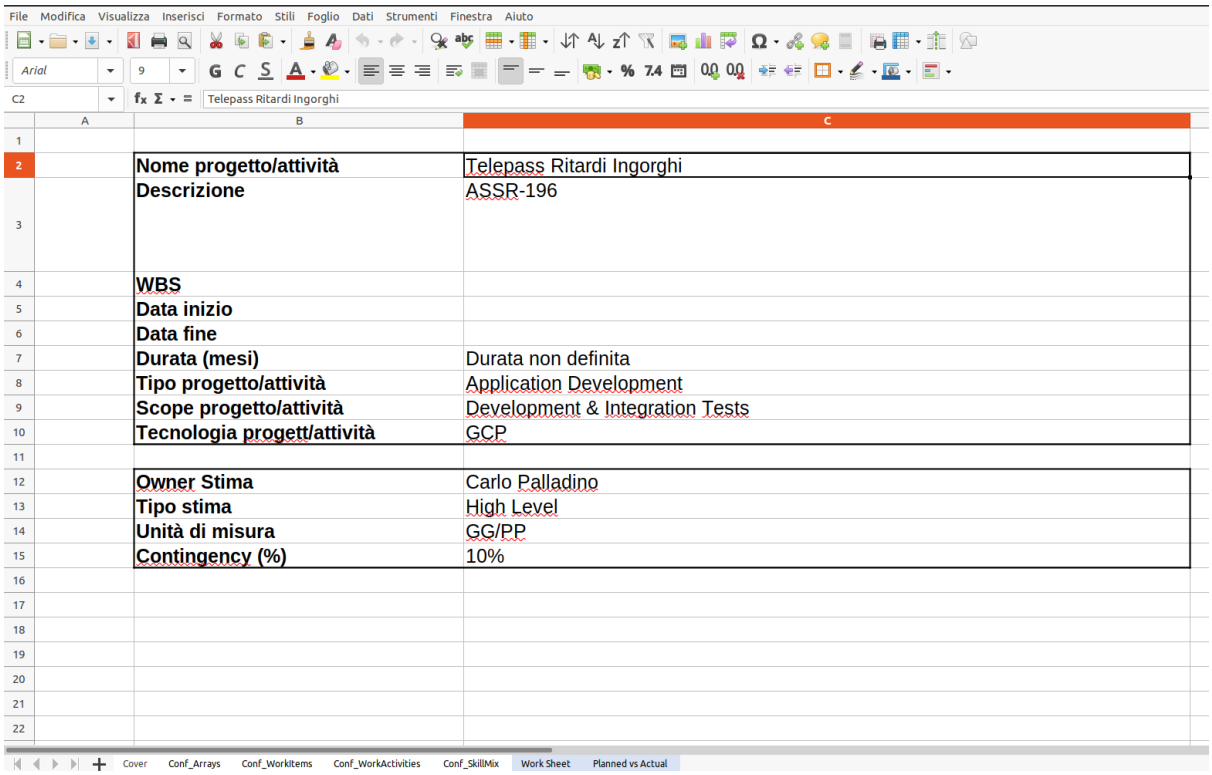
3. Requisiti di business

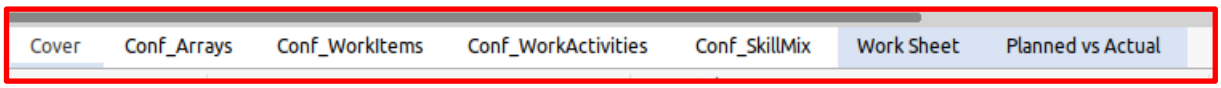
3.1 Esigenze

Come accennato precedentemente, i tipi di strumenti CMMI variano in base alle esigenze aziendali.

Quello che ho preso in esame riguarda il calcolo delle stime.

Ad oggi in NTT Data per le stime CMMI è stato messo a disposizione un documento Excel:





Il documento è diviso in sette fogli:

1. Cover, che è una rappresentazione parziale della stima finale;
2. Conf_Arrays, che contiene informazioni essenziali (divise in tabelle), per il calcolo della stima come unità di misura, tipo di stima, tipo di progetto, scopo del progetto, tecnologia del progetto, oggetto da lavoro, complessità, release, fase, requisiti e costo;

3. Conf_WorkItems, che è la rappresentazione del work item (oggetto da lavoro) rapportato alla sua complessità e di altre informazioni;
4. Conf_WorkActivities, che ci fornisce dati relativi alle varie attività;
5. Conf_SkillMix, che da informazioni dei costi di ogni lavoro rapportati con i costi delle singole figure lavorative e della loro percentuale di allocazione nel progetto;
6. WorkSheet, che contiene informazioni più complete delle stime;
7. Planned vs Actual, dove si confrontano le pianificazioni con i dati attuali e reali.

Un grande problema è che differenti gruppi di lavoro utilizzano differenti fogli excel per modellare le stesse informazioni necessarie al CMMI. In aggiunta, una recente attività di certificazione CMMI ha reso necessaria la raccolta di ulteriori e diverse informazioni rispetto a quelle che si raccoglievano prima.

3.2 Obiettivi

Si aspira, seguendo le esigenze aziendali, alla creazione di un software, nello specifico una web app, che sostituisca quello che si può considerare uno strumento obsoleto, per quanto riguarda il calcolo delle stime, per il suo uso non intuitivo. Si ambisce a realizzare uno strumento che permetta la condivisione e la modifica di dati online tra più utenti: come già specificato, sul mercato sono presenti molti strumenti relativi al CMMI, alcuni dei quali optano per una gestione di "limited sharing" delle informazioni cioè che permettono all' utente di condividere i propri documenti con altri user ma non permette a quest'ultimi di modificarli. Questo tipo di condivisione, attuato ad esempio dal sopracitato SoftExpert sul software Excellence Suite, permette una buona protezione dei dati ma non permette la compartecipazione del lavoro tramite web. Le operazioni sulle informazioni e successivamente anche il calcolo delle stime devono essere controllati oltre che protetti: risulta quindi fondamentale implementare un login collegato ad un database, già preesistente, contenente le credenziali di accesso. L'emigrazione della grande mole di informazioni del suddetto db potrebbe diventare una grossa problematica se si optasse per un software già esistente.

Un altro obiettivo da raggiungere, come già menzionato, è quello che l'utilizzo dell'app sia per un'utenza generica, per questo un ottimo software come quello della Compliance Power 365, CMMI 365, non è stato ritenuto consono al tipo di uso che si è prestabilito dello strumento da realizzare e non idoneo poiché accessibile esclusivamente ad un'utenza specializzata.

Oltretutto, la maggior parte dei software che si occupano di CMMI sono proprietari, quindi realizzare internamente un nuovo CMMI estimator rappresenta una soluzione a costo zero.

Questi sono gli obiettivi determinati, i punti discussi giustificano il perché non è stato preso ad esame un software già presente sul mercato come quelli esplicitati precedentemente.

L'utilizzo di tale strumento ha fini autovalutativi per lavori interni al contesto aziendale affinché restituisca in maniera preventiva se essi rispettano o meno i canoni per eventuali certificazioni CMMI.

4. Requisiti software

4.1 Architettura monolitica

L'architettura monolitica rappresenta il tradizionale modello per lo sviluppo di un software ed è caratterizzata dal fatto di contenere tutte le funzioni all'interno di un unico blocco.
Un software definito dall'architettura monolitica è logicamente autonomo, in quanto i suoi componenti sono fortemente accoppiati e interdipendenti. In altri termini, i componenti di un software monolitico non possono essere separati o riutilizzati senza procedere con l'aggiornamento integrale dell'intera applicazione. Una modifica puntuale comporta pertanto la ricompilazione dell'intero codice per creare un nuovo eseguibile. Non è possibile disaccoppiare une singola funzione e procedere soltanto al suo aggiornamento.
Questo può rappresentare in alcuni casi uno svantaggio.
La scelta di utilizzare questo tipo di architettura, l'ho fatta considerando che un software monolitico è caratterizzato da una serie di vantaggi intrinseci nella sua architettura informatica, che ho ritenuto consoni per il progetto che sono andato a realizzare.

4.1.1 Test e Debug

Il fatto di contenere tutti i componenti all'interno di un'unica applicazione fa sì che le architetture monolitiche si prestino in maniera più rapida ed efficiente alle fasi di test e alle operazioni di debug del software, in quanto tutto rimane sotto lo stretto controllo dello sviluppatore. La forte dipendenza tra i componenti stessi rende evidente in maniera immediata qualsiasi problema di funzionamento, dal momento che si manifesta in maniera immediata nel funzionamento dell'applicazione.

4.1.2 Design

Se le metodologie di sviluppo di moderna concezione, come DevOps e Agile, risultano particolarmente efficienti nel caso di applicazioni vaste, complesse e caratterizzate da un ciclo di vita lungo ed articolato, che non possono prescindere da una logica modulare, l'architettura monolitica continua a rappresentare la metodologia di design più semplice ed immediata per concepire un'applicazione capace di eseguire simultaneamente più funzioni.
Il monolitico, per la sua stessa natura, non presenta rischi di "sovra-architettura", in quanto tutto il necessario per far funzionare il software è contenuto all'interno dell'applicazione stessa, senza dover ricorrere ad una complessa architettura di componenti tra loro disaccoppiati, che possono certamente risultare più comodi nelle fasi di sviluppo, ma vanno successivamente messi in comunicazione tra loro per consentire il completo funzionamento del software.
Nel caso dell'architettura monolitica questo sforzo non è necessario. Per tali ragioni, lo sviluppo da zero di un'applicazione monolitica, specie se comparata con altre metodologie, risulta piuttosto rapida, in quanto il suo design non prevede la complessità di comunicazione tra i componenti chiamati a dialogare con l'interfaccia principale.

4.1.3 Performance

La struttura monolitica del software riduce ai minimi termini la comunicazione interna tra i componenti, rendendo l'esecuzione più rapida, soprattutto al primo avvio, quando ancora non è stato effettuato un caching dell'applicazione nella memoria di sistema: caching che comporta a sua volta un'ulteriore complessità da gestire a livello di sviluppo.
Quando si prevede di sviluppare un software che usato simultaneamente da persone che non necessitano di un gran numero di funzioni, o un software rimarrà confinato all'interno di una singola linea di business, l'architettura monolitica può generare evidenti vantaggi a livello di performance, che si configurano in maniera sinergica con gli altri aspetti positivi sin qui rilevati.

4.1.4 Sicurezza

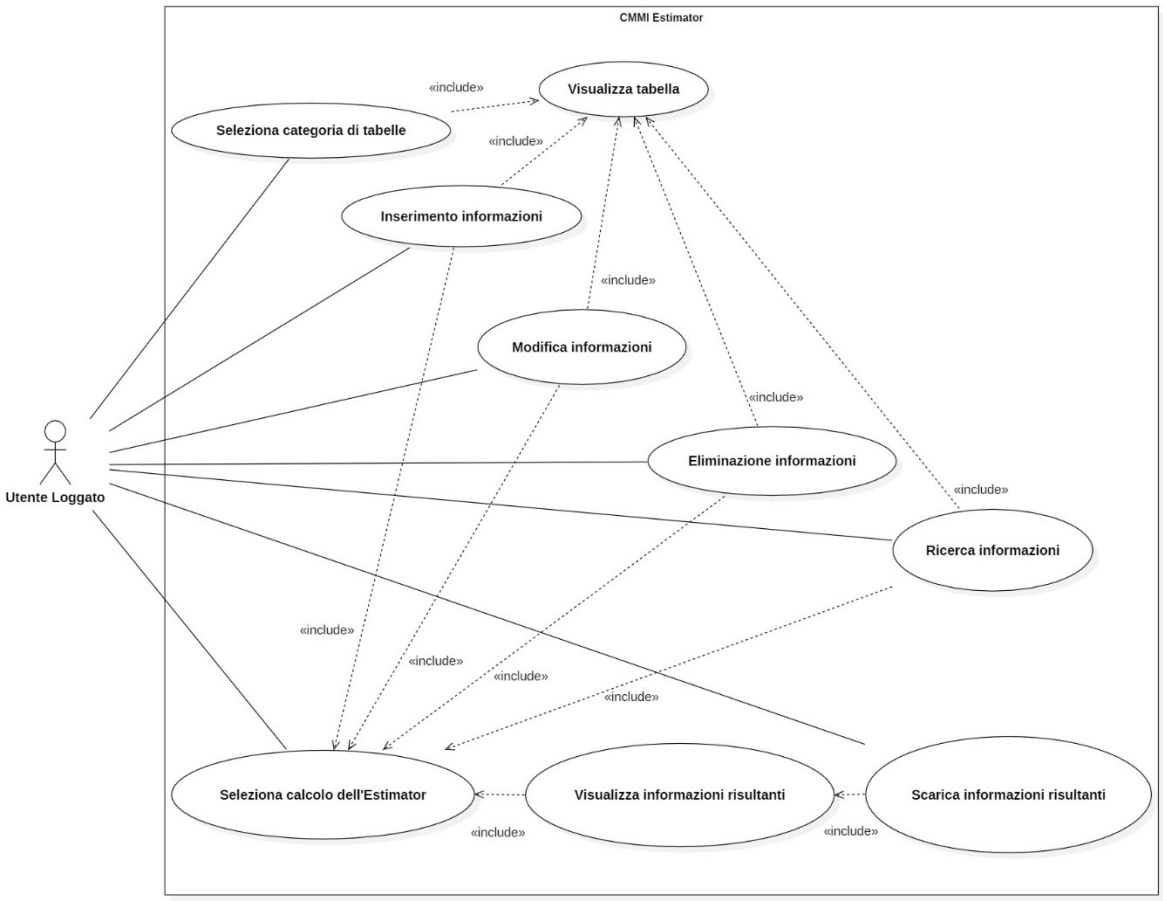
Disporre di un'applicazione concentrata in un unico blocco agevola notevolmente il controllo delle autorizzazioni sugli accessi e sulla possibilità di modificare nei modi più svariati gli elementi che la compongono. Per applicazioni di simile dimensioni e complessità, la superficie di attacco di un'architettura monolitica si presenta piuttosto ridotta, se confrontata con le architetture basate su componenti disaccoppiati come ad esempio nel caso dei microservizi. Un ulteriore vantaggio è costituito dall'utilizzo di un database centralizzato, particolare che rende a sua volta più semplici le procedure relative alla protezione dei dati, tra cui il backup e ripristino in caso di incidenti.

5. Considerazioni e soluzioni tecnologiche

5.1 Progettazione logica dell'applicativo

Si procede con l'esplicitazione di quella che sarà la classe di utenza che utilizzerà il sistema, Use Case diagram e tabelle di Cockburn, rispettivamente, per fornire una rappresentazione dell'interfaccia del sistema da usare come riferimento per le descrizioni strutturate, definire gli attori e i casi d'uso previsti nell'utilizzo dell'applicativo e per evidenziare eventuali criticità nello sviluppo di un caso d'uso e gestirne le situazioni di errore. La classe di utenza alla quale si affaccia la web application è quella formata da tutti i dipendenti della azienda che mi ha ospitato per il tirocinio, quindi utenti registrati che hanno come credenziali di accesso e-mail aziendale e password. Inoltre, si procede nel rappresentare le varie relazioni tra le tabelle ricavate dal documento Excel fornitomi, riproducendo un modello entità-relazione.

5.1.1 Use case diagram



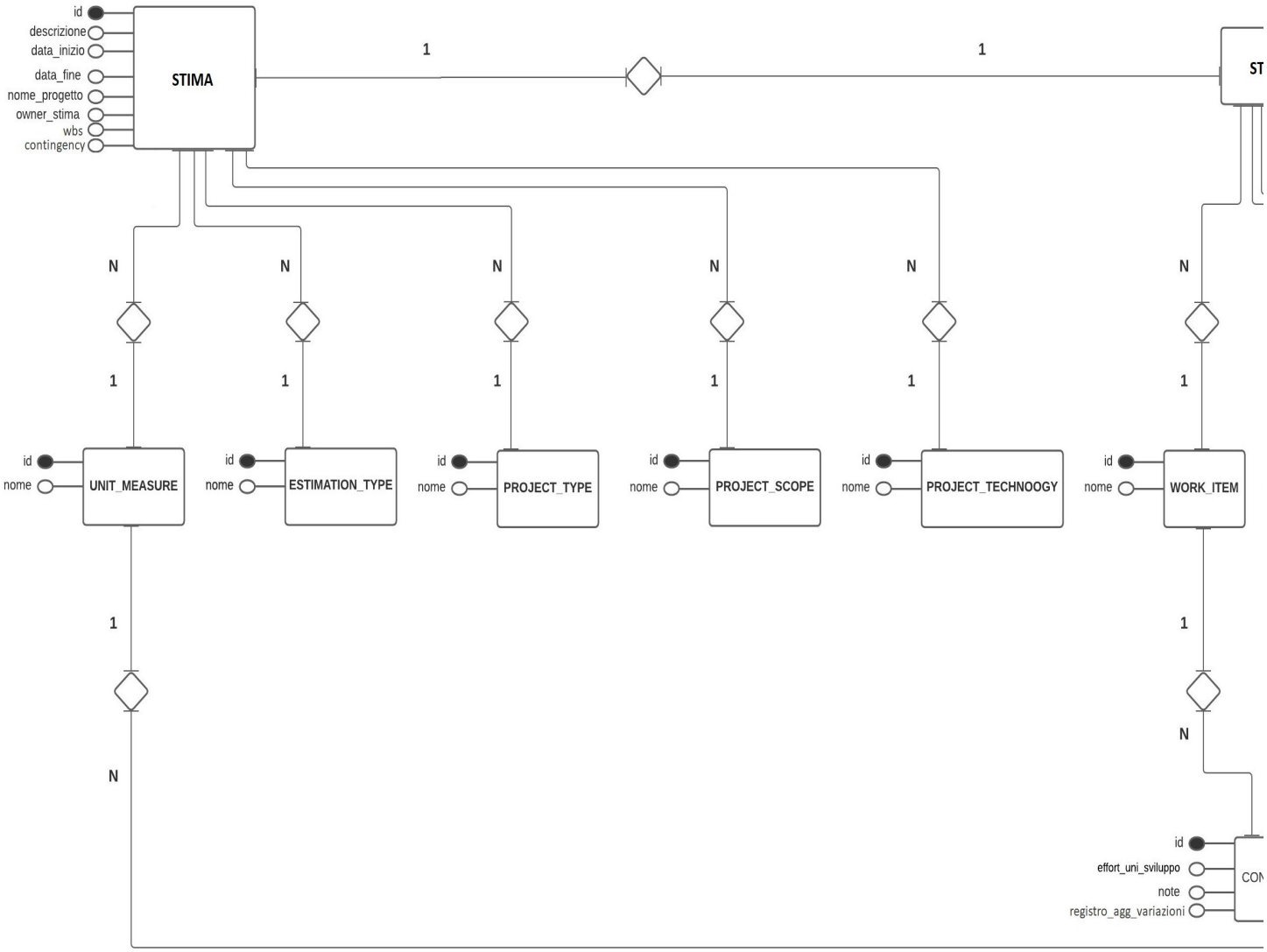
5.1.2 Tabelle di Cockburn

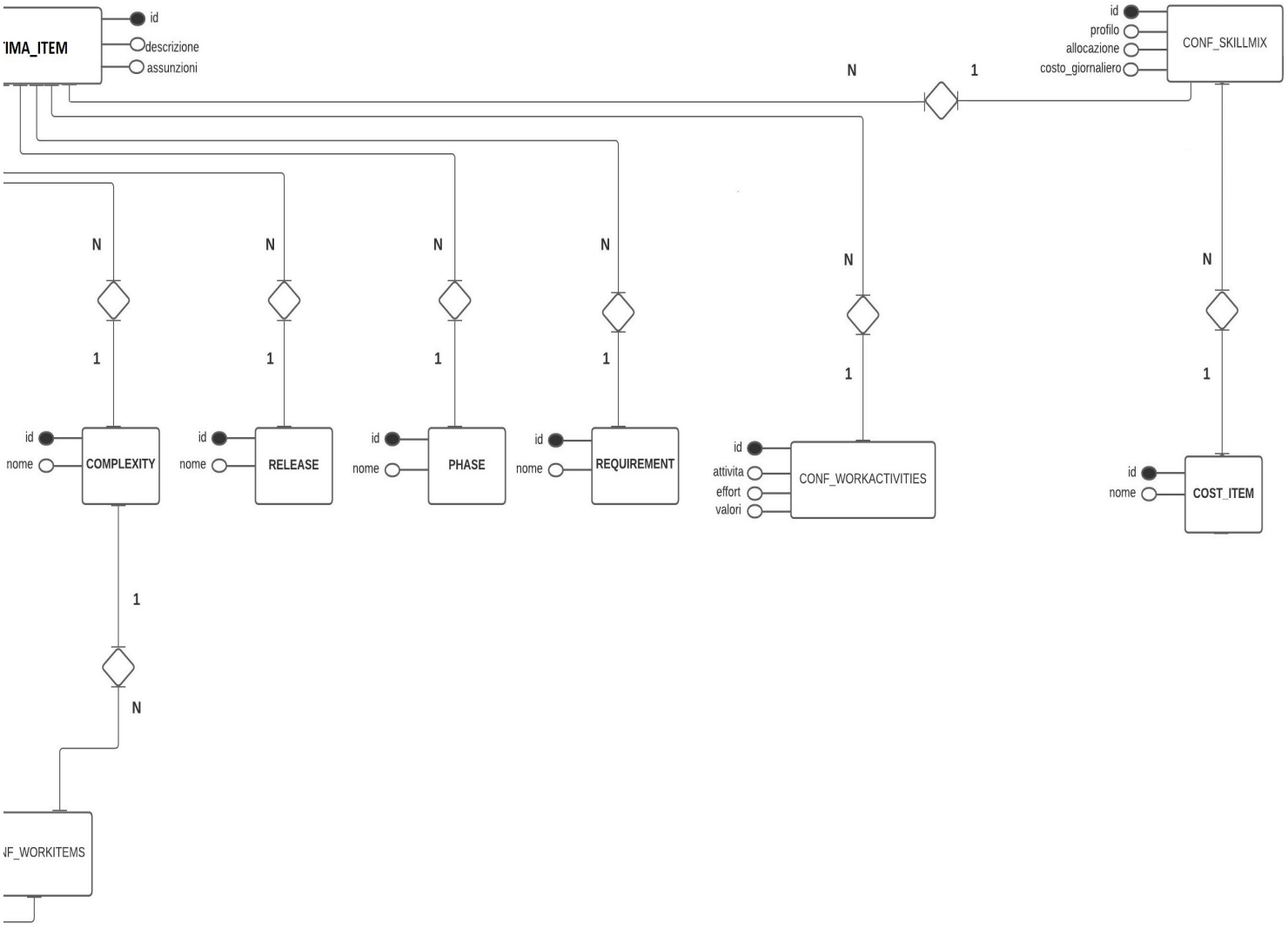
USE CASE #1	Aggiungere una stima.		
Goal in context	L'utente riesce ad inserire una stima.		
Scope e level	• L'utente deve aver effettuato il login; • Va sulla pagina dell'estimator.		
Success end condition	Viene aggiunta una nuova stima.		
Failed end condition	Non viene aggiunta la stima.		
Primary, secondary actors	Utente loggato.		
Trigger	Clicca su "Insert new stima".		
Description	**Step**	**Utente loggato**	**Sistema**
	1	Clicca su "Insert new stima"	
	2		Mostra finestra inserimento.
	3	Inserisce il nome del progetto	
	4	Inserisce la descrizione	
	5	Inserisce la data di inizio	
	6	Inserisce la data di fine	
	7	Inserisce owner stima	
	8	Inserisce contingency	
	9	Inserisce il tipo della stima	
	10	Inserisce scope del progetto	
	11	Inserisce unità di misura	
	12	Inserisce tecnologia del progetto	
	13	Inserisce il tipo del progetto	
	14	Clicca su "Save"	
	15		Mostra alert "Success".
	16		Mostra tabella aggiornata.
Extections	**Step**	**Branching Action**	
<Salva omettendo informazioni>	3.1	Clicca su "Save"	
	3.2		Mostra alert "Error".

USE CASE #2	Eliminazione di una stima.		
Goal in context	L'utente riesce ad eliminare una stima.		
Scope e level	• L'utente deve aver effettuato il login; • Va sulla pagina dell'estimator.		
Success end condition	Viene eliminata una stima.		
Failed end condition	Non viene aliminata la stima.		
Primary, secondary actors	Utente loggato.		
Trigger	Clicca sul bottone delete.		
Description	**Step**	**Utente loggato**	**Sistema**
	1	Clicca sul bottone delete.	
	2		Mostra alert "Success".
Extections	**Step**	**Branching Action**	
<La riga non può essere eliminata poiché collegata a Stimaitem>	3.1	Clicca sul bottone delete.	
	3.2		Mostra alert "Error".

USE CASE #3	Ricerca di un nome della tabella.		
Goal in context	L'utente riesce ad eliminare una stima.		
Scope e level	• L'utente deve aver effettuato il login; • Va sulla pagina di Confarrays. • Seleziona la tabella.		
Success end condition	Viene trovata la riga		
Failed end condition	Non viene aliminata la stima.		
Primary, secondary actors	Utente loggato.		
Trigger	Clicca sul bottone search.		
Description	**Step**	**Utente loggato**	**Sistema**
	1	Clicca sul bottone search.	
	2		Mostra finestra inserimento.
	3	Inserisce il nome da ricercare.	
	4	Clicca sul bottone "Send"	
	5		Mostra la tabella con l'elemento ricercato.
Extections	**Step**	**Branching Action**	
<Utente non inserisce il nome>	3.1	Clicca sul bottone "Send"	
	3.2		Mostra tabella vuota.

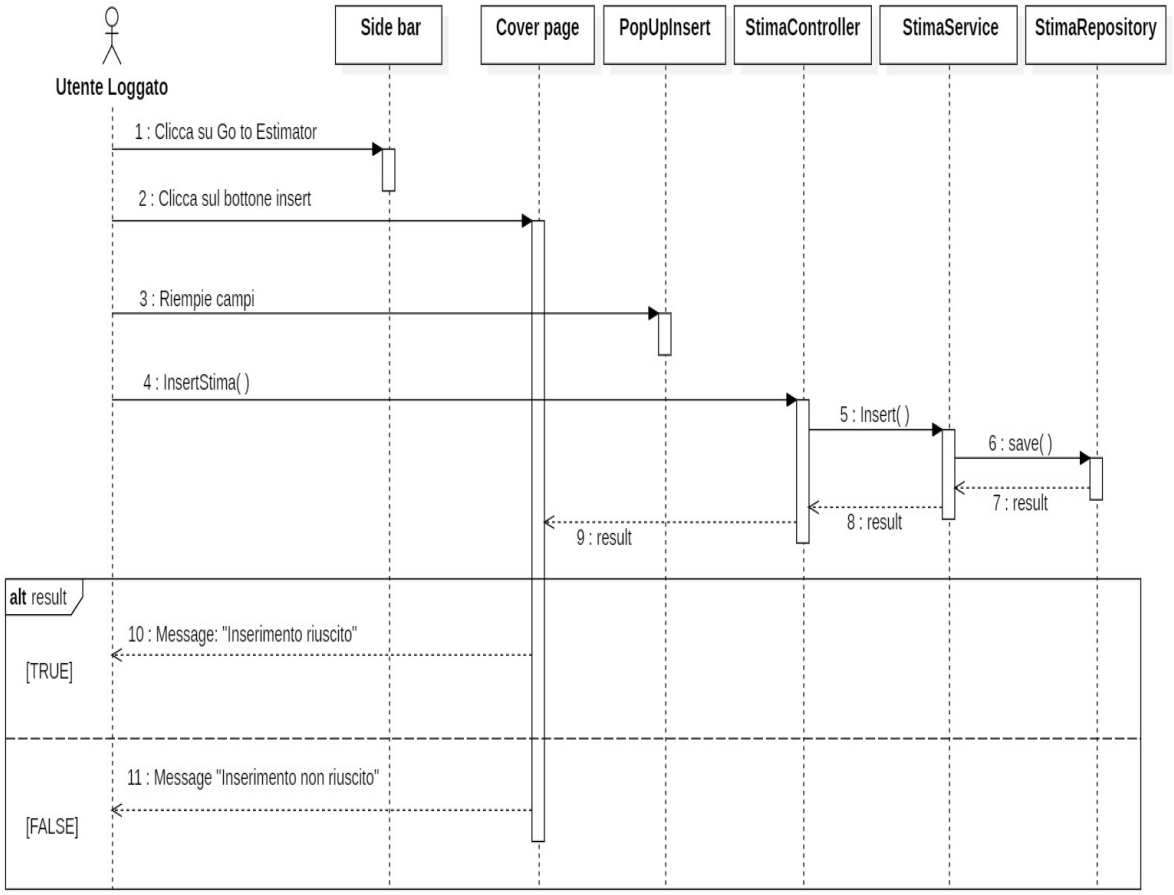
5.1.3 E-R


id
descrizione
data_inizio
data_fine
nome_progetto
owner_stima
wbs
contingency
STIMA
1
1
ST
N
N
N
N
N
N
1
1
1
1
1
1
id
nome
UNIT_MEASURE
id
nome
ESTIMATION_TYPE
id
nome
PROJECT_TYPE
id
nome
PROJECT_SCOPE
id
nome
PROJECT_TECHNOOGY
id
nome
WORK_ITEM
1
1
N
N
id
effort_uni_sviluppo
note
registro_agg_variazioni


IMA_ITEM
id
descrizione
assunzioni
id
profilo
allocazione
costo_giornaliero
CONF_SKILLMIX
N
1
N
1
N
1
N
1
N
1
N
1
id
nome
COMPLEXITY
id
nome
RELEASE
id
nome
PHASE
id
nome
REQUIREMENT
id
attivita
effort
valori
CONF_WORKACTIVITIES
id
nome
COST_ITEM
1
N
NF_WORKITEMS

5.1.4 Sequence diagram

Il sequence diagram rappresentato, descrive il funzionamento dell'applicazione nel momento in cui l'utente loggato decide di inserire una nuova stima.



5.2 Architettura

Nell'era digitale in cui viviamo, le app hanno eletto come loro principale dimora gli smartphone e i tablet, ma è possibile trovarle anche sui computer desktop. Nel caso delle app native si tratta di software creati su misura per una piattaforma, mentre le web app, come tipologia di applicazione da me realizzata, funzionano tramite browser e si differenziano dalle app native per alcune caratteristiche.

Le applicazioni web non possono essere adattate perfettamente all'hardware del dispositivo in uso, ma funzionano su tutti i sistemi operativi e su tutti i dispositivi, che dispongono di uno dei browser supportati dalle web app. In teoria, quindi, basta un'unica app per riuscire ad essere presenti su tutte le piattaforme, anche se non è sempre possibile ottimizzarla per tutti i browser.

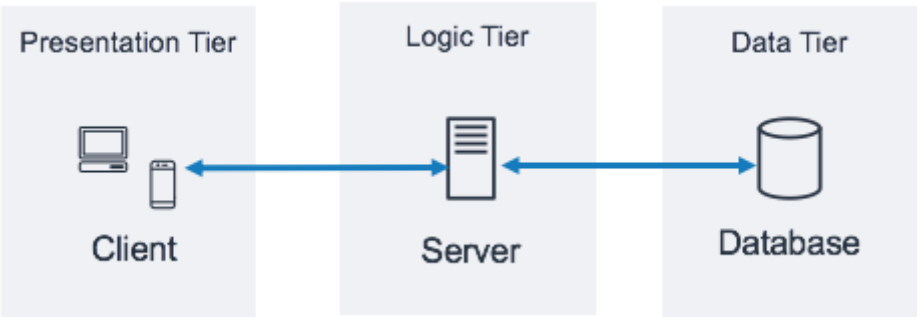
Un ulteriore punto a favore risiede nella risoluzione delle falle di sicurezza: nel caso delle app native si possono risolvere solo scaricando una nuova versione o un aggiornamento, mentre in un'applicazione web vengono apportati i dovuti aggiornamenti di sicurezza direttamente sul software, in modo che tutti gli utenti accedano automaticamente alla versione più sicura. Inoltre, i costi complessivi per le web app sono inferiori e vengono create più velocemente.

Un'applicazione web è composta essenzialmente da due parti:

- una "invisibile" all'esterno, "motore" di tutto il resto, senza la quale il sito non funzionerebbe detta back-end;
- una visibile, quella con cui interagisce l'utente, chiamato frontend.

L'architettura che ho deciso di utilizzare è quella chiamata three-tier architecture. L'architettura a tre livelli è un'architettura applicativa software consolidata che organizza le applicazioni in tre livelli di elaborazione logici e fisici:

1. Il livello di presentazione o interfaccia utente;
2. Il livello dell'applicazione, in cui i dati sono trattati;
3. Il livello dati, in cui vengono archiviati e gestiti i dati associati all'applicazione.



Il principale vantaggio dell'architettura a tre livelli è che, poiché ogni livello viene eseguito sulla propria infrastruttura, ogni livello può essere sviluppato simultaneamente da un team di sviluppo separato e può essere aggiornato o ridimensionato in base alle esigenze senza influire sugli altri livelli.

Sia app che interfaccia web inviano e ricevono all'interno dei messaggi http oggetti di tipo JSON, l'approccio utilizzato è di tipo REST. La scelta è dettata proprio dalla necessità futura di poter permettere alla parte logica di poter comunicare con diverse rappresentazioni.

5.2.1 Data tier

La prima parte del backend che andrò a descrivere è quella riguardante il database.

Il database del CMMI Estimator sarà di tipo relazionale. La mia scelta di RDBMS (Relational Database Management System) è ricaduta su PostgreSQL. Esso è un sistema di gestione di database open source. Rispetto ad altri RDBMS, PostgreSQL si distingue per il suo supporto alle categoriche ed integraliste funzionalità dei database orientati agli oggetti e/o relazionali, ad esempio il supporto completo per le transazioni sicure, cioè Atomicità, Consistenza, Isolamento, Durabilità (ACID), ed è completamente compatibile con le proprietà di supporto di chiave esterna, trigger e il metodo unione.

ACID deriva dall'acronimo inglese Atomicity, Consistency, Isolation, e Durability. Sono alcune proprietà che un DBMS dovrebbe sempre garantire per ogni transazione.

La versione di PostgreSQL su cui ho lavorato è la 14.1 .PostgreSQL mette a disposizione un'interfaccia grafica, da me usata, che prende il nome di PgAdmin. PgAdmin è un'applicazione multipiattaforma, scritta in C++, che consente di amministrare in modo semplificato.

Il database dell'estimatore è composto da sedici tabelle:

- 5 che rappresentano le informazioni contenute nei fogli Cover, Conf_Workitem, Conf_WorkActivities, ConfSkillMix, WorkSheet (che nel database prenderà il nome di StimaItem) del documento excel;
- 11 che rappresentano le tabelle contenute nel foglio che ha il nome di Conf_Arrays.

Per quanto riguarda il login, mi è stato fornito un database pre implementato e popolato da NTT Data. Quindi l'application lavora su due DB: il primo che immagazzina tutti i dati relativi al login e il secondo che contiene tutte le informazioni riguardanti l'estimator.

5.2.1.1 Dizionario dei dati

Classe	Attributi
complexity	● id (integer): rappresenta l'identificatore della singola riga (singola informazione); ● nome (character varying(50)): rappresenta il tipo di complessità.
cost_item	● id (integer): rappresenta l'identificatore della singola riga (singola informazione); ● nome (character varying(50)): rappresenta il tipo di cost item.
estimation_type	● id (integer): rappresenta l'identificatore della singola riga (singola informazione); ● nome (character varying(50)): rappresenta il tipo di stima.
phase	● id (integer): rappresenta l'identificatore della singola riga(singola informazione); ● nome (character varying(50)): contiene il valore della fase.
project_scope	● id (integer): rappresenta l'identificatore della singola riga (singola informazione); ● nome (character varying(50)): contiene il nome dello scopo del progetto.
project_technology	● id (integer): rappresenta l'identificatore della singola riga (singola informazione); ● nome (character varying(50)): contiene il nome della tecnologia del progetto.

project_type	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • nome (character varying(50)): contiene il nome del tipo di progetto.
release	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • nome (character varying(50)): contiene il nome del release.
requirement	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • nome (character varying(50)): contiene il nome del requisito.
unit_measure	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • nome (character varying(50)): contiene il nome dell'unità di misura.
work_item	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • nome (character varying(50)): contiene il nome del work item.
conf_skillmix	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • profilo(character varying(50)): contiene il ruolo del dipendente. • allocazione (float(2)): contiene il valore dell'allocazione.
conf_workactivities	• id (integer): rappresenta l'identificatore

	della singola riga (singola informazione); • attivita (character varying(250)): contiene il tipo di attività. • effort (float(2)): contiene il valore dell'effort. • valori_pred (float(2)): contiene valori predefiniti.
conf_workitem	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • effort_uni_sviluppo (float(2)): contiene il valore dell'effort unitario di sviluppo. • note (character varying(250)): contiene le note riguardanti la stima. • registro_agg_variazioni (integer): contiene i valori del registro aggiustamento variazioni.
stima	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • nome_progetto (character varying(100)): contiene il nome del progetto. • wbs (character varying(250)): contiene la stringa rappresentante la struttura wbs usata. • descrizione (character varying(250)): contiene la descrizione della stima. • data_inizio (date): contiene la data di inizio del calcolo della stima.

	• data_fine (date): contiene la data di fine del calcolo della stima. • owner_stima (character varying(100)): contiene il valore del owner stima. • contingency (float(2)): contiene il valore di contigenza.
stimaitem	• id (integer): rappresenta l'identificatore della singola riga (singola informazione); • descrizione (character varying(250)): contiene la descrizione del worksheet. • assunzioni (character varying(250)): contiene le assunzioni fatte

5.2.1.2 Progettazione logica DB

Nota: In grassetto sono evidenziate le entità, con il grassetto e sottolineatura le **Primary key** e con la sottolineatura e il *corsivo* le *foreign key*

COMPLEXITY (**id**, nome)

COST_ITEM (**id**, nome)

ESTIMATION_TYPE (**id**, nome)

PROJECT_TECHNOLOGY (**id**, nome)

PROJECT_TYPE (**id**, nome)

PROJECT_SCOPE (**id**, nome)

RELEASE (**id**, nome)

REQUIREMENT (**id**, nome)

PHASE (**id**, nome)

UNIT_MEASURE (**id**, nome)

WORK_ITEM (**id**, nome)

CONF_SKILLMIX (**id**, profilo, allocazione, costo_giornaliero, *id_cost_item*)

Foreign key:
id_cost_item: riferito all'attributo id (primary key) della classe **COST_ITEM**.

CONF_WORKACTIVITIES (**id**, attivita, effort, valori_pred)

CONF_WORKITEMS (**id**, effort_uni_sviluppo, note, registro_agg_variazioni, *id_unit_measure*, *id_work_item*, *id_complexity*)

Foreign keys:
id_unit_measure: riferito all'attributo id (primary key) della classe **UNIT_MEASURE**;
id_work_item: riferito all'attributo id (primary key) della classe **WORK_ITEM**;
id_complexity: riferito all'attributo id (primary key) della classe **COMPLEXITY**.

STIMA (**id**, nome_progetto, wbs, descrizione, data_inizio, data_fine, owner_stima, contingency, *id_estimationtype*, *id_project_scope*, *id_unit_measure*, *id_project_technology*)

Foreign keys:
id_estimationtype: riferito all'attributo id (primary key) della classe **ESTIMATION_TYPE**;
id_project_scope: riferito all'attributo id (primary key) della classe **PROJECT_SCOPE**;

id_unit_measure: riferito all'attributo id (primary key) della classe **UNIT_MEASURE**;
id_project_scope: riferito all'attributo id (primary key) della classe **PROJECT_SCOPE**.

STIMA_ITEM (id, descrizione, assunzioni, *id_release, id_phase*, *id_requirement*, *id_work_item*, *id_complexity, id_conf_workactivities, id_conf_skillmix, id_stima*)

Foreign keys:
id_release: riferito all'attributo id (primary key) della classe **RELEASE** ;
id_phase: riferito all'attributo id (primary key) della classe **PHASE** ;
id_requirement: riferito all'attributo id (primary key) della classe **REQUIREMENT** ;
id_work_item: riferito all'attributo id (primary key) della classe **WORK_ITEM** ;
id_complexity: riferito all'attributo id (primary key) della classe **COMPLEXITY** ;
id_conf_workactivities: riferito all'attributo id (primary key) della classe **CONF_WORKACTIVITIES** ;
id_conf_skillmix: riferito all'attributo id (primary key) della classe **CONF_SKILLMIX** ;
id_stima: riferito all'attributo id (primary key) della classe **STIMA**.

5.2.1.3 Definizione schema logico (implementazione delle tabelle e dei domini)

complexity:

```
CREATE TABLE IF NOT EXISTS complexity
(
    id integer NOT NULL,
    nome character varying(50)  NOT NULL UNIQUE,
    CONSTRAINT id_complexity PRIMARY KEY (id)
);

ALTER TABLE IF EXISTS complexity
OWNER to postgres;
```

Al campo nome sono associate i constraint NOT NULL e UNIQUE poiché l'attributo non può avere valore NULL e non possono essere presenti complessità con lo stesso nome (lo stesso vale per il resto delle tabelle contenute in Conf_Arrays).

Nota: Verranno omesse le altre tabelle contenute nel foglio Conf_Arrays tra le create table poichè c'è similarità tra le strutture delle tabelle rappresentate in quel foglio.

conf_workactivities:

```
CREATE TABLE IF NOT EXISTS conf_workactivities
(
    id integer NOT NULL,
    attivita character varying(250) NOT NULL UNIQUE,
    effort float(2) DEFAULT 0 CHECK(effort>=0 AND effort<=100),
    valori_pred      float(2) DEFAULT 0 CHECK(valori_pred>=0 AND valori_pred<=100),
    CONSTRAINT id_conf_workactivities PRIMARY KEY (id)
);

ALTER TABLE IF EXISTS conf_workactivities
OWNER to postgres;
```

Il CHECK è un vincolo viene utilizzato per limitare l'intervallo di valori che può essere inserito in una colonna. Se si definisce un CHECK su una colonna, verranno consentiti solo determinati valori per quella colonna.

Se si definisce un CHECK su una tabella, è possibile limitare i valori in alcune colonne in base ai valori in altre colonne nella riga.

Il CHECK sui campi effort e valori_pred, sta ad indicare che possono avere solo valori che stanno nel range [0,100]. Oltretutto a quest'ultimi è assegnato per default (nel caso non siano inseriti) il valore 0.

conf_workitems:

```
CREATE TABLE IF NOT EXISTS conf_workitem
(
    id integer NOT NULL,effort_uni_sviluppo float(2),
    note character varying(250),
    registro_agg_variazioni integer,
    id_unit_measure integer NOT NULL,
    id_work_item integer NOT NULL,
    id_complexity integer NOT NULL,

    CONSTRAINT id_conf_workitems PRIMARY KEY (id),
    FOREIGN KEY(id_unit_measure) REFERENCES unit_measure(id),
    FOREIGN KEY(id_work_item) REFERENCES work_item(id),
    FOREIGN KEY(id_complexity) REFERENCES complexity(id),
);

ALTER TABLE IF EXISTS conf_workitems
OWNER to postgres;
```

conf_skillmix:

```
CREATE TABLE IF NOT EXISTS conf_skillmix
(
    id integer NOT NULL,
    profilo character varying(50) NOT NULL UNIQUE,
    allocazione float(2) DEFAULT 0 CHECK(allocazione>=0 AND allocazione<=100),
    costo_giornaliero float(2),
    id_cost_item integer NOT NULL,


    CONSTRAINT id_conf_skillmix PRIMARY KEY (id),
    FOREIGN KEY(id_cost_item) REFERENCES cost_item(id)
);

ALTER TABLE IF EXISTS conf_skillmix
OWNER to postgres;
```

conf_workitems:

```
CREATE TABLE IF NOT EXISTS conf_workitems
(
    id integer NOT NULL,
    effort_uni_sviluppo float(2),
    note character varying(250),
    registro_agg_variazioni character varying(250),
    id_unit_measure integer NOT NULL,
    id_work_item integer NOT NULL,
    id_complexity integer NOT NULL,


    CONSTRAINT id_conf_workitems PRIMARY KEY (id),
    FOREIGN KEY(id_unit_measure) REFERENCES unit_measure(id),
    FOREIGN KEY(id_work_item) REFERENCES work_item(id),
    FOREIGN KEY(id_complexity) REFERENCES complexity(id)

);

ALTER TABLE IF EXISTS worksheet
OWNER to postgres;

ALTER TABLE conf_workitems
ADD CONSTRAINT uni_work_complexity UNIQUE(id_complexity, id_work_item);
```

Le chiavi esterne di complexity e work_item insieme sono un vincolo di unicità cioè non possono essere presenti righe in conf_workitems con coppie uguali di queste foreign keys. Questo è motivato dal fatto che nella tabella conf_workitems, che servirà per il calcolo di alcuni risultati e per la visualizzazione dell'effort, a ogni work item dovrà corrispondere una sola complessità.

stima:

```
CREATE TABLE IF NOT EXISTS stima
(
    id  integer NOT NULL,
```

```
    nome_progetto character varying(100),
    wbs character varying(250),
    descrizione character varying(250),
    data_inizio date,
    data_fine date,
    owner_stima character varying(100),
    contingency float(2),
    id_estimationtype integer NOT NULL,
    id_project_scope integer NOT NULL,
    id_unit_measure integer NOT NULL,
    id_project_type integer NOT NULL,
    id_project_technology integer NOT NULL,

    CONSTRAINT id_stima PRIMARY KEY (id),
    FOREIGN KEY(id_estimationtype) REFERENCES estimation_type(id),
    FOREIGN KEY(id_project_scope) REFERENCES project_scope(id),
    FOREIGN KEY(id_unit_measure) REFERENCES unit_measure(id),
    FOREIGN KEY(id_project_technology) REFERENCES project_technology(id),
    FOREIGN KEY(id_project_type) REFERENCES project_type(id)

);

ALTER TABLE IF EXISTS stima
OWNER to postgres;

ALTER TABLE stima
ADD CONSTRAINT CHK_stimaDate CHECK (data_inizio<=data_fine);
```

Nella tabella stima vige la restrizione che la data di inizio delle stime deve essere minore della data di fine, per coerenza temporale.

stima_item:

```
CREATE TABLE IF NOT EXISTS stima_item
(
    id integer NOT NULL,
```

```
    descrizione character varying(250),
    assunzioni character varying(250),
    id_release integer NOT NULL,
    id_phase integer NOT NULL,
    id_requirement integer NOT NULL,
    id_work_item integer NOT NULL,
    id_complexity integer NOT NULL,
    id_workactivities integer NOT NULL,
    id_stima integer NOT NULL,
    id_conf_skillmix integer NOT NULL,


    CONSTRAINT id_worksheet PRIMARY KEY (id),
    FOREIGN KEY(id_release) REFERENCES release(id),
    FOREIGN KEY(id_phase) REFERENCES phase(id),
    FOREIGN KEY(id_requirement) REFERENCES requirement(id),
    FOREIGN KEY(id_work_item) REFERENCES work_item(id),
    FOREIGN KEY(id_complexity) REFERENCES complexity(id),
    FOREIGN KEY(id_workactivities) REFERENCES conf_workactivities(id),
    FOREIGN KEY(id_stima) REFERENCES stima(id),
    FOREIGN KEY(id_conf_skillmix) REFERENCES conf_skillmix(id)
);

ALTER TABLE IF EXISTS stima_item
OWNER to postgres;
```

Per la gestione delle cancellazioni delle chiavi primarie, ho optato per la politica ON DELETE NO ACTION.
ON DELETE NO ACTION specifica che se si tenta di eliminare una riga contenente una chiave a cui fanno riferimento chiavi esterne in righe esistenti in altre tabelle, queste eliminazioni non saranno permesse. Non è stato necessario specificare questo comportamento poiché, almeno che non venga specificata una gestione diversa, questa è la gestione predefinita dell'eliminazioni delle chiavi primarie di postgreSQL.

5.2.2 Logic tier

Lo strumento utilizzato per lo sviluppo del logic tier è Eclipse che è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma, ed è inoltre un software libero.

Il linguaggio usato per il livello logico è Java (versione 8). La scelta è ricaduta su java essenzialmente per 2 motivi:

1. Per la sua popolarità nel mondo informatico;
2. Per la sua portabilità.

In generale, si dice portabile un linguaggio che consente di scrivere programmi in grado di funzionare correttamente su piattaforme hardware diverse e sotto differenti sistemi operativi, richiedendo semplicemente la ricompilazione dei sorgenti nel nuovo ambiente (e dunque, implicitamente, con una differente implementazione del compilatore).

Al fine di semplificare il lavoro con Java sono stati sviluppati negli ultimi decenni diversi framework che forniscono all'utente una struttura di base pronta all'uso per lo sviluppo di programmi Java. Per questo motivo, per la realizzazione del logic tier, il codice java è stato supportato da diversi frameworks:

- Spring Boot;
- Spring Data;
- Spring Security.

Spring Boot è il framework per creare applicazioni basate sul framework Java Spring che sono subito pronte per ambienti di produzione e sono maggiormente utilizzate per creare microservizi.

Offre i seguenti vantaggi:

- configurazione Maven semplificata grazie ai POM "Starter" (Project Object Models);
- configurazione automatica di Spring, ove possibile;
- fornitura di caratteristiche non funzionali come metriche o configurazioni esternalizzate.

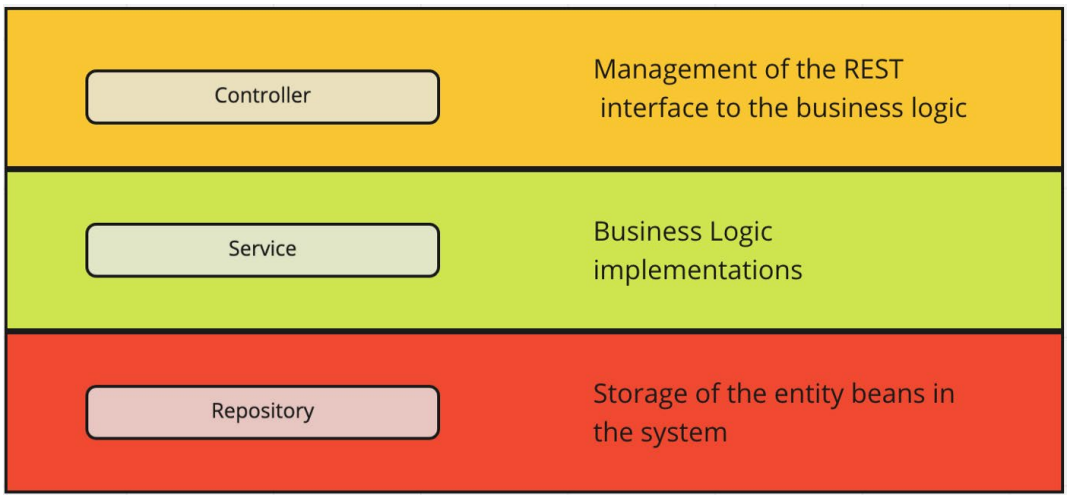
Spring Data è un progetto SpringSource di alto livello il cui scopo è unificare e facilitare l'accesso a diversi tipi di archivi di persistenza, sia sistemi di database relazionali che archivi di dati NoSQL.

Spring Data framework nato con l'obiettivo di fornire un modello standard per l'implementazione del data access layer all'interno di un progetto Spring.

Spring Data definisce un set di API indipendente dalla sorgente dati utilizzata, fornendo l'implementazione di queste interfacce per molte delle più diffuse tecnologie di persistenza dei dati. Per questo progetto è stato usato Spring Data JPA, che implementa la specifica JPA tramite Hibernate e ci consente di lavorare con database relazioni.

Spring Security fornisce autenticazione, autorizzazione e altre funzionalità di sicurezza per le applicazioni. Il suo utilizzo è fondamentale per la parte del codice riguardante la gestione del login.

Il pattern utilizzato nell'applicazione è del tipo Controller-Service-Repository.



Il modello Controller-Service-Repository ha un utilizzo dominante in molte applicazioni Spring Boot. Uno dei grandi motivi per utilizzare questo modello è che fa un ottimo lavoro di separazione delle preoccupazioni:

- Controller: contiene la logica dell'applicazione e trasmette i dati di input dell'utente al service;

- Service: è il middleware tra controller e repository, raccoglie i dati dal controller, esegue la convalida e la logic business e chiama i repository per la manipolazione dei dati;
- Repository: livello per l'interazione con i modelli e l'esecuzione di operazioni DB.

Inoltre, è presente anche il Model (o entity) che rappresenta l'entità contenuta nel database.

La granularità di questi livelli è questa:

- per ogni tabella DB esiste un model;
- per ogni modello esiste un repository;
- per ogni repository c'è un servizio;
- per ogni servizio c'è un controller.

Per ogni model sono presenti due classi che si occupano della gestione dei dati (vengono utilizzati solo per trasferire dati da un processo o contesto a un altro):

- Classe Eto, che si occupa dei dati semplici;
- Classe Cto, che si occupa dei dati complessi.

Eto	**Cto**

public class ComplexityEto {
 	
private Long id;
 	
private String nome;
 	
….
}
	public class ComplexityCto {
	
	private Long id;
	
	private String nome;
	
	private List<StimaItemEto>
	items;
	
	/* avendo StimaItem come chiave esterna l'id di Complexity */

	private List<ConfWorkItemsEto> confworkitems; /* avendo ConfWorkItems come chiave esterna l'id di Complexity */

Per ogni Entità sono stati implementati metodi che permettono:

- Insert di nuovi elementi;
- Update delle informazioni del singolo elemento;
- Search del singolo elemento in base ad un'informazione presa in input;
- Delete del singolo elemento in base ad un'informazione presa in input.

5.2.2.1 Esempi struttura

Sono mostrati esempi di implementazione di metodi della struttura precedentemente descritti nel dettaglio.

Controller:

```
public static final Logger logger = LoggerFactory.getLogger(ComplexityController.class);

@Autowired
private ComplexityService complexityService;

/**
 * I commenti inseriti in questo Controller sono attuabili anche negl'altri
 * poichè la struttura dei metodi è pressochè identica.
 */

/**
 * Ricerca tutti gli elementi e ritorna un ResponseEntity
 *
 * @return
 */
@GetMapping("/all")
public ResponseEntity<List<ComplexityCto>> findByComplexity() {

        logger.info("Sono nel Controller findAll, complexity ({})");

        /**
         * Richiamo del metodo getComplexity().
         */
        List<ComplexityCto> ris = complexityService.getComplexity();

        if (ris == null) {
                return new ResponseEntity<List<ComplexityCto>>(ris, HttpStatus.INTERNAL_SERVER_ERROR);
                /**
                 * Ritorna "500 Internal Server Error" se il valore di ritorno di
                 * getComplexity() è uguale a NULL.
                 */

        } else {
                return new ResponseEntity<List<ComplexityCto>>(ris, HttpStatus.OK);
                /**
                 * Ritorna "200 OK" se il valore di ritorno di getComplexity() è diverso da
                 * NULL.
                 */
        }
}
```

Service:

```
public List<ComplexityCto> getComplexity() {

        /**
         * L' oggetto Logger viene utilizzato per registrare i messaggi per un sistema
         * specifico o un componente dell'applicazione.
         */
        logger.info("findAll Complexity.");

        List<ComplexityCto> complexity;

        ModelMapper mapper = new ModelMapper();

        /**
         * Richiamo del metodo findAll().
         */
        List<ComplexityEntity> ris = complexityRepository.findAll();

        /**
         * Gestione delle eccezioni.
         */
        try {
                /**
                 * Quando il metodo map viene chiamato, i tipi di origine (source) e di
                 * destinazione (destination) vengono analizzati per determinare quali proprietà
                 * corrispondono implicitamente in base a una strategia di corrispondenza e ad
                 * altre configurazioni . I dati vengono quindi mappati in base a queste
                 * corrispondenze.
                 */
                complexity = mapper.map(ris, listComplexityCto);

                /**
                 * INFO message: le informazioni sono per l'uso di amministratori o utenti
                 * avanzati. Indica principalmente le azioni che hanno portato a un cambiamento
                 * di stato per l'applicazione.
                 */
                logger.info("Complexity mostrata correttamente ({})", complexity);

        } catch (Exception e) {

                /**
                 * La Logging Java errors è un componente essenziale che ci consente di
                 * tracciare la causa principale degli errori. Per impostazione predefinita, i
                 * messaggi di registro Java vengono archiviati solo sulla console.
                 */
                logger.error("Errore durante il caricamento del complexity, ({})", e.getMessage());
                complexity = null;
        }

        return complexity;
}
```

Repository:

```
/**
 * Ritorna tutte le righe della tabella Complexity.
 *
 * @param
 * @return
 */
List<ComplexityEntity> findAll();
```

Per la maggior parte dei metodi, nel repository non è stato necessario specificare una query nativa. Questo perchè Spring Boot ha come predefinite alcune operazioni di base come la ricerca per un determinato elemento (findBy), la ricerca di una determinata tabella (find), l'inserimento e l'aggiornamento (save) e l'eliminazione di un determinato elemento (delete) o in base al valore di un determinato campo (deleteBy).

Importante in questo livello è stata l'aggiunta della class CORSFilter. Essa non permette il verificarsi di errori relativi al CORS riscontrabili nel front-end.

5.2.3 Presentation tier

Per la realizzazione di questo livello è stato fondamentale l'uso Angular (versione 11), un framework open source per lo sviluppo di applicazioni web e l'editor utilizzato è stato Visual Studio code. Il linguaggio di programmazione usato per Angular è Typescript.
La motivazione che sta dietro alla scelta di questo framework è che Angular è stato progettato per fornire uno strumento facile e veloce per sviluppare applicazioni che girano su qualunque piattaforma inclusi smartphone e tablet. Infatti le applicazioni web in Angular in combinazione con dei toolkit, nel caso specifico di questo progetto Bootstrap (versione 4.6.0), diventano responsive, ossia il design del sito web si adatta in funzione alle dimensioni del dispositivo utilizzato.
Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS.

A supporto per Bootstrap, sono stati utilizzati Popper.js, una libreria utilizzata per gestire i popper nelle applicazioni web, e jQuery una libreria basata su Javascript che viene usata per creare applicazioni single page.

Fondamentale per la struttura dell'intero applicativo, più nello specifico per la comunicazione tra front-end e back-end, la definizione degli endpoints e del network.service .

```
export const ENDPOINT = {
  appRoot : environment.appRoot,

  login: environment.authServer + 'oauth/token',

  //Config Complexity
  confComplexityGetAll : environment.appRoot + "complexity/all",
  confComplexityGetNome : environment.appRoot + "complexity/search",
  confComplexitySave : environment.appRoot + "complexity/save",
  confComplexityDelete : environment.appRoot + "complexity/delete",
```

(fig. 1)

```
export const environment = {
  production: false,
  appRoot:"http://localhost:8080/",
  authServer:"http://localhost:8081/"
};
```

(fig. 2)

Alla fig. 1 vengono mostrati al fine di esempio gli endpoints relativi al component Login e dei metodi del component Complexity, si noti che la definizione degli endpoints è diversa.
Per spiegare il perchè, ci viene in aiuto la fig. 2: Da quest'ultima si evidenzia quello detto al paragrafo 5.1.1 cioè che il login lavora su un database diverso rispetto a quello degli altri components.

Nel network.service vengono definite una get generica, una post generica e il login.

```
login(username: string, password: string) {

    let headers: HttpHeaders = new HttpHeaders();
    headers = this.buildAuthHeader();

    let parametri: FormData = new FormData();
    parametri.append("username", username);
    parametri.append("password", password);

    return this.http.post('http://localhost:8081/oauth/token?grant_type=password', parametri,
    {
        headers: headers, withCredentials: false, }).toPromise().then(response => {

        localStorage.setItem('userToken', JSON.parse(JSON.stringify(response)).access_token); //setto il localStorage
        console.log("LocalStorage ---" + JSON.stringify(localStorage.getItem('userToken')));

        return JSON.stringify(response);//response restituisce token

    }).catch(error => {
        console.error('An error occurred', error);
        return Promise.reject(error.message || error);
    });
}

private buildAuthHeader(): HttpHeaders {

    var headers = new HttpHeaders({
        "Authorization": "Basic " + btoa('prova:prova')
    });

    return headers;

}
```

La funzione di login, oltre a permettere accesso o meno da parte di un utente, restituisce all'user un token, che viene salvato nel localstorage, che concede i permessi per effettuare le varie operazioni sulle tabelle.

```
private buildHeader(): HttpHeaders {

    var headers = new HttpHeaders({
        "Authorization": "Bearer " + localStorage.getItem("userToken"),
        "X-Auth-Token": ""+localStorage.getItem("userToken")
    });
    return headers;
}
```

Grazie a questa funzione, che viene richiamata da get e post generiche, il token viene aggiunto all'intestazione HTTP (così da fornire i permessi al back-end).

```
//Post generica
postRequest(endpoint: string, body: any) {
    let headers: HttpHeaders = new HttpHeaders();
    headers = this.buildHeader();

    console.log("POST on " + endpoint);

    return this.http
        .post(endpoint, body, { headers: headers })
        .toPromise()
        .then(response => {
            return JSON.parse(JSON.stringify(response));
        })
        .catch(error => {
            console.error('An error occurred', error);
            if (error.status == 401) {
                this.router.navigate(['/login']);
            }
            return Promise.reject(error.message || error);
        });
}
```

```
//Get Generica
getRequest(endpoint: string) {

    let headers: HttpHeaders = new HttpHeaders();
    headers = this.buildHeader();

    console.log("HEADERS " + JSON.stringify(headers));
    console.log("GET on " + endpoint);

    return this.http.get(endpoint, { headers: headers , withCredentials: true}).toPromise().then(response => {
        return JSON.parse(JSON.stringify(response));
    }).catch(error => {
        console.error("An error occurred", error);
        if (error.status == 401) {
            this.router.navigate(['/login']);
        }
        return Promise.reject(error.message || error);
    })
}
```

Per quanto riguarda la get e la post, queste vengono richiamate nei components per rievocare le varie funzionalità, precedentemente descritte, dal back-end.

```
/**
 * Inserimento elementi nella tabella Complexity.
 * @param complexity
 */
save(complexity:string){

  console.log("Sono qui --"+this.complexity);

  this.ns.postRequest(ENDPOINT.confComplexitySave,{nome: complexity}).then(response => {
  console.log(JSON.stringify(response));
  this.comp = response;
  this.getcomplexity();
  this.showSuccess("Inserimento effettuato.");
   }).catch( error => {
     console.log("Errore durante la stampa della tabella. ");
     this.showError("Inserimento non effettuato!");
   })
}
```

```
/**
 * Ricerca nella tabella Complexity in base al campo nome.
 * @param searchStr
 */
search(searchStr:string){
 console.log("Sono qui --"+this.searchStr);

 this.ns.getRequest(ENDPOINT.confComplexityGetNome+"/"+searchStr).then(response => {
  console.log(JSON.stringify(response));

  this.listComplexity = [response];
  }).catch( error => {
     console.log("Errore durante la stampa della tabella. "+error);
     this.showError("Ricerca non effettuata!");
  })

}
```

Per ogni tabella presente nel database è stata dichiarata un'entity in angular e è stato realizzato un component che richiama i metodi dal logic tier e fa visualizzare all'utente le informazioni contenute nel data tier; quindi, comunicano esclusivamente su endpoints riferiti alle operazioni della propria tabella di appartenenza.

Altro component fondamentale ai fini strutturali del front-end sono quelli relativi alla gestione dell'error 404 (page not found) che ha il compito di informare l'utente dell'errore e fornire un link per far ritornare l'user all'homepage.

Alcuni components come Confskillmix, per esigenze grafiche necessarie per all'interazione con l'utente, lavorano anche su endpoints appartenenti ad altre tabelle.

Esempio: per il riempimento di un select element per deve permettere la selezione di un elemento di tipo Costitem nella pagina html dedicata a Confskillmix, è stato necessario richiamare il metodo della stampa della tabella Costitem e di conseguenza è stato necessario lavorare su un endpoint dedicato a quest'ultima, per permettere la visualizzazione dei suoi contenuti.

Questo tipo di approccio è stato usato con molta frequenza in tabelle aventi chiavi esterne, per permettere le associazioni di valori alle foreign key nelle fasi di insert o update delle righe di queste tabelle.

Per il calcolo di alcuni attributi non è stato necessario interpellare il back-end.

Tra questi:

- La somma del valore delle allocazioni e la somma di prodotti visualizzata nella pagina dedicata a Confskillmix;

```
getsumAllocazioni(){
  this.sum=0;

  for (let confskillmix of this.listConfskillmix)
  {
    this.sum= this.sum+confskillmix.allocazione
  }
}

getsumproduct(){
  this.sumproduct=0;

  for (let confskillmix of this.listConfskillmix)
  {
    this.sumproduct= this.sumproduct+((confskillmix.allocazione/100)*confskillmix.costogiornaliero);
  }
}
```

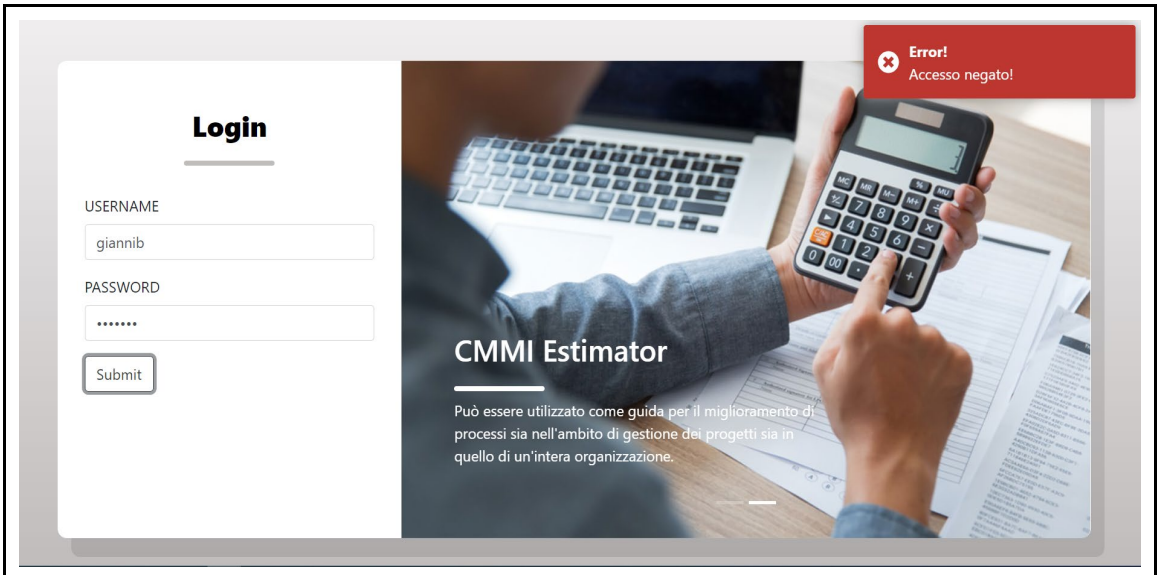
- Il calcolo della durata visualizzata nella tabella della pagina dedicata a Stima.

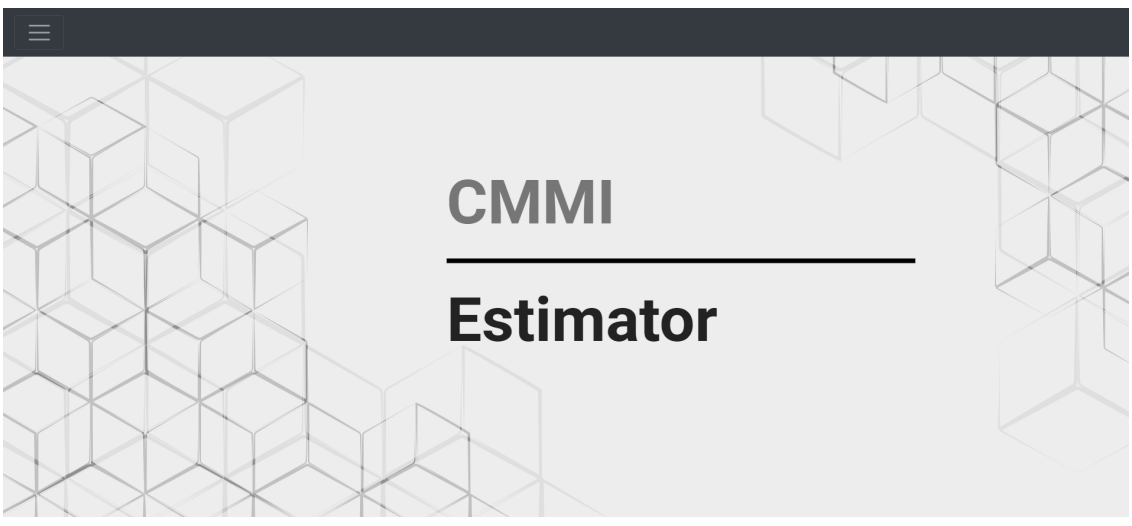
```
/**
 * Fa la differenza tra le date e genera il campo "Durata" che è presente nella tabella a video
 * @param data1
 * @param data2
 */
calculateDiff(data1:Date,data2:Date){
  let date1 = new Date(data1);
  let date2 = new Date(data2);
  this.days=0;

  this.days = Math.floor((date1.getTime() - date2.getTime()) / 1000 / 60 / 60/ 24);
}
```

6. Descrizione delle funzionalità

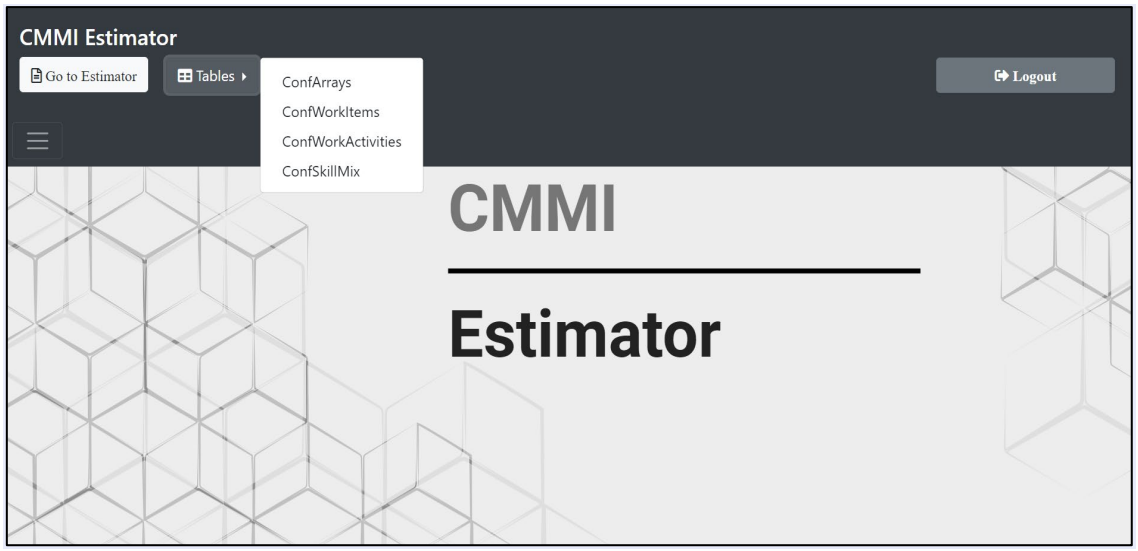
L'obiettivo prefissato è stato quello di realizzare un'application semplice, intuitiva e che inoltre tenga sempre informato l'utente dell'esito delle sue operazioni.

La prima pagina dell'applicazione ad apparire è quella del login. Da notare l'alert in alto a destra, la presenza di questi sarà una costante quando gli users faranno operazioni per il motivo precedentemente spiegato.
Effettuato il login con successo si presenterà l'homepage.

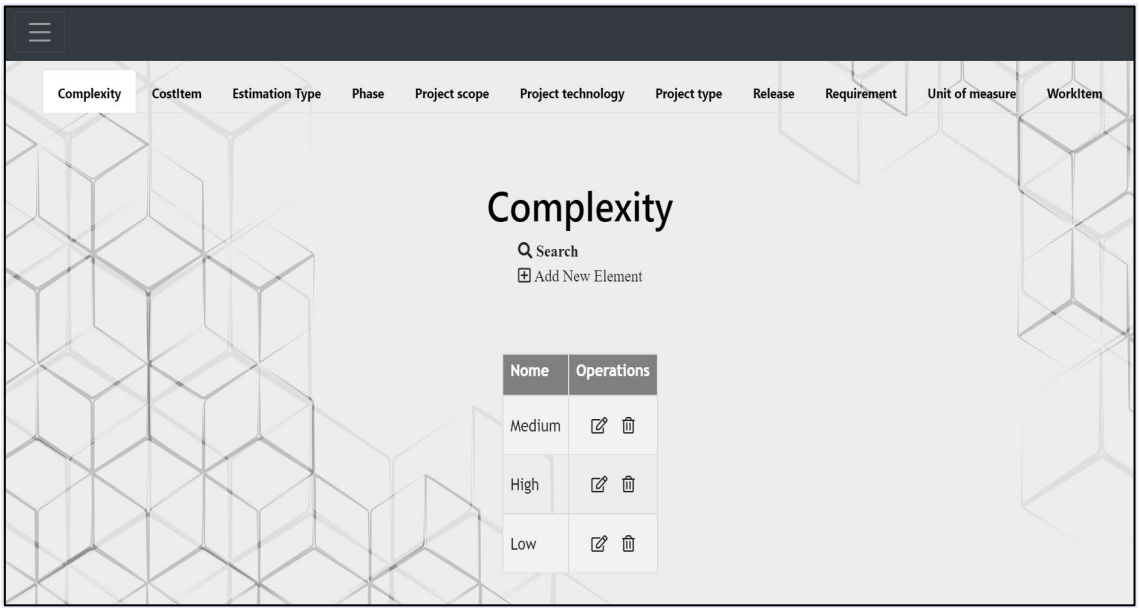




Caratteristica di quest'application la presenza fissa di una side bar che è il mezzo per spostaci da una pagina all'altra.
Cliccando sulla side bar, essa si "abbassa" ci vengono mostrate tutte le varie opzioni di navigazione. La divisione delle tabelle è similare al documento exel preso in esame e la struttura delle tabelle nelle varie pagine è simile tra loro.

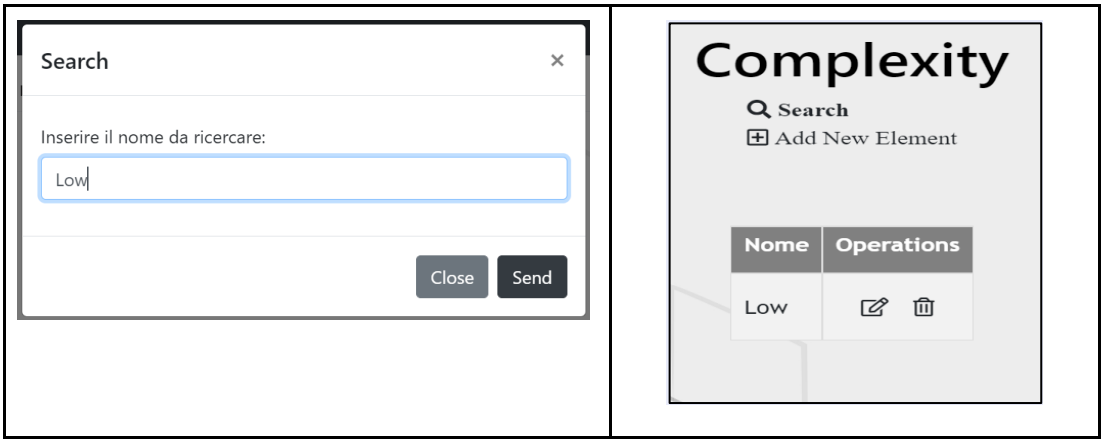


Prendiamo in esame la pagina ConfArrays: si evidenzia la presenza di una nav-tab. La nav-tab permette di selezionare e visualizzare tutte le tabelle contenute in questa pagina.

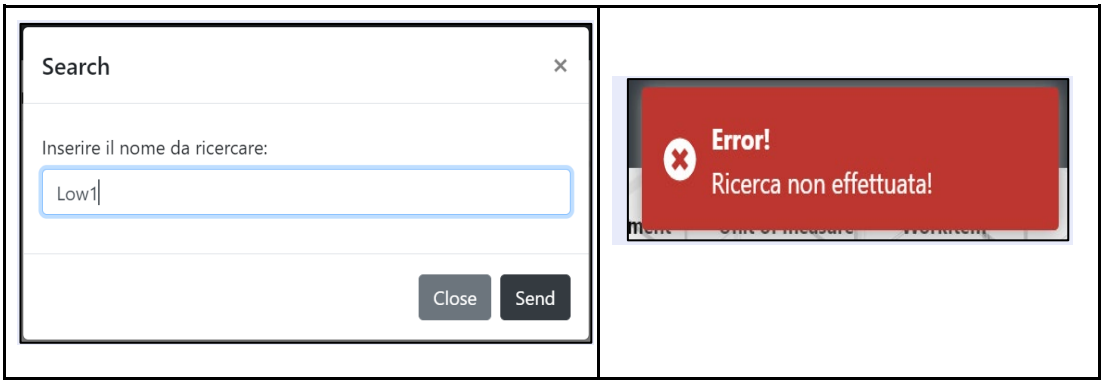


Cliccando su una delle varie opzioni visualizzeremo, oltre al contenuto della singola tabella anche le varie operazioni che si possono effettuare su queste.
Abbiamo a disposizione l'operazione di ricerca e quella di inserimento, cioè operazioni attuabili sull'intera tabella, che sono mostrate esternamente alla struttura tabellare.
Sono state aggiunte anche le operazioni di modifica e cancellazione, cioè operazioni possibili sulla singola riga della tabella, che vengono mostrate riga per riga in un campo della struttura tabellare che prende il nome di "Operations".

Cliccando sull'operazione di ricerca, verrà mostrato un modal.
Esso permetterà l'inserimento di una parola che sarà poi ricercata tra le informazioni presenti.

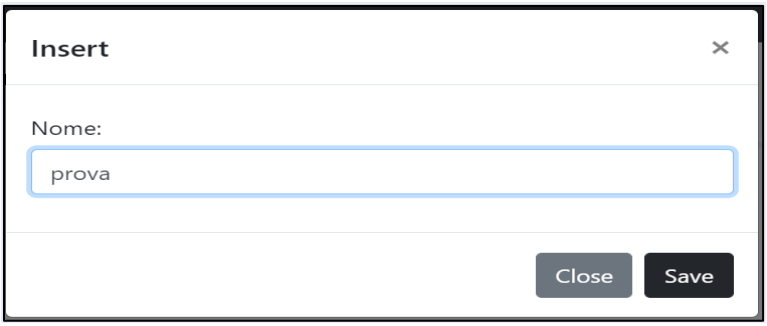


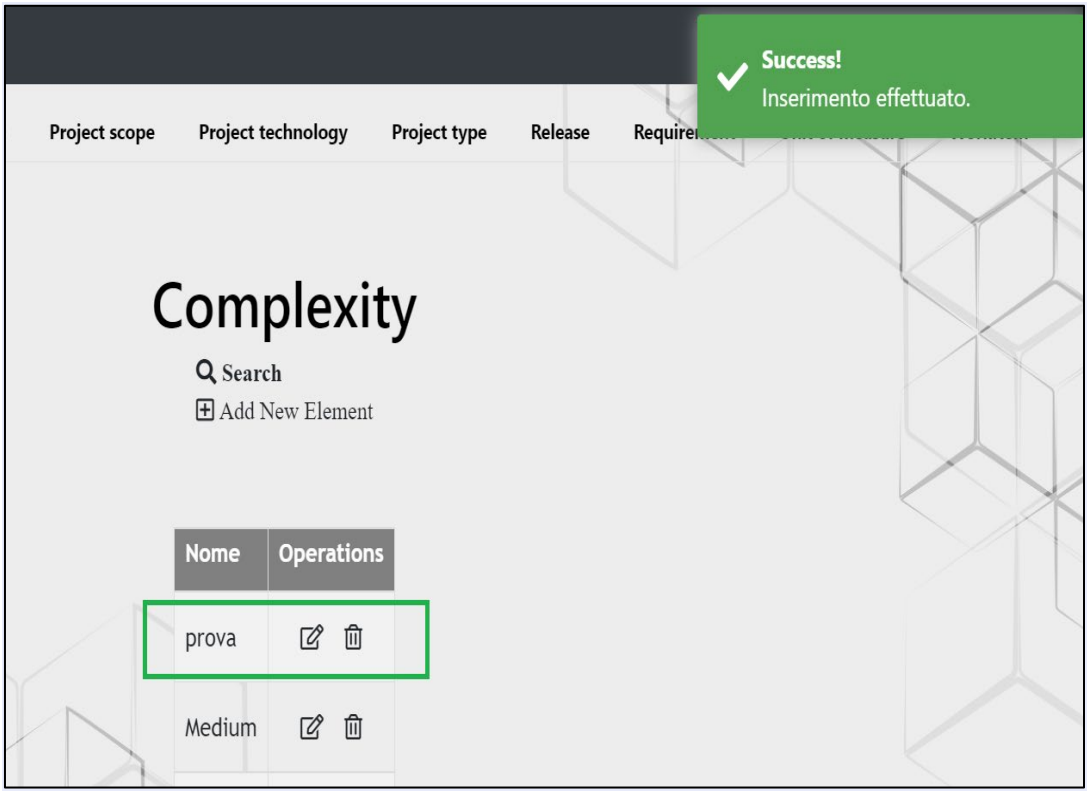
Nel caso la riga sia presente verrà mostrata nella tabella e verrà mostrato un'alert che informerà l'utente della buona riuscita dell'operazione. Se ciò non avviene l'alert informerà di un errore.

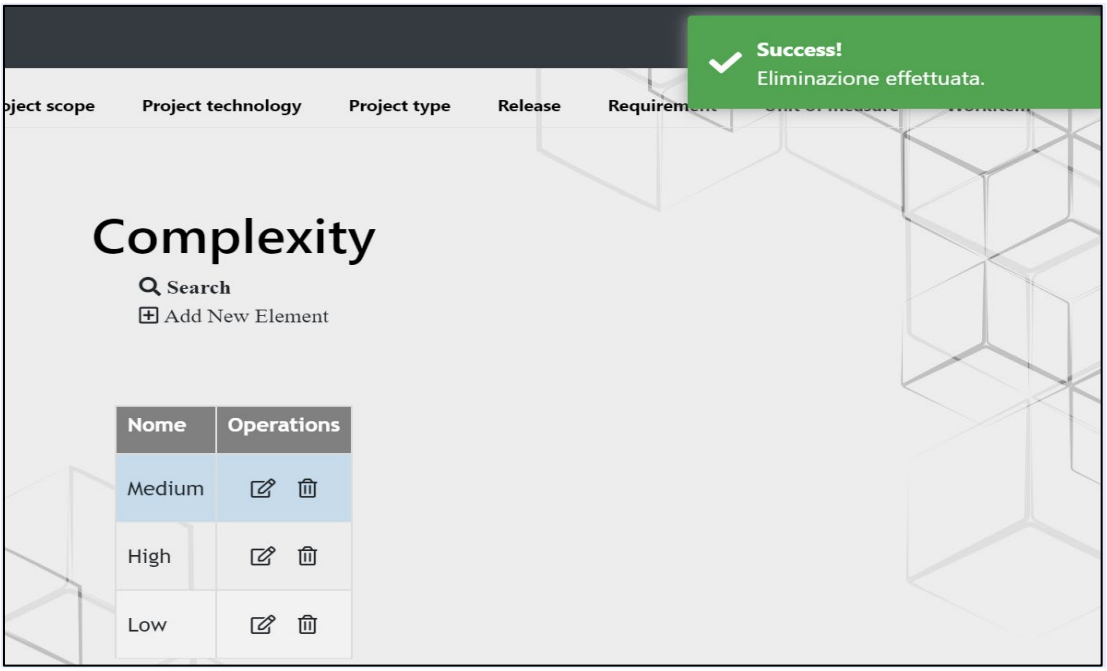


Lo stesso discorso va fatto per le operazioni di inserimento, modifica e cancellazione.

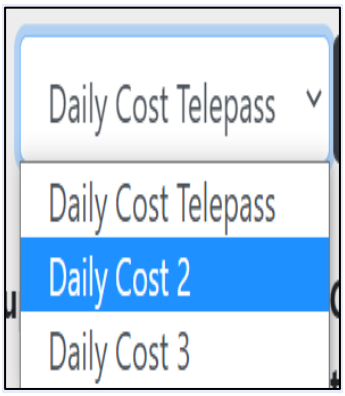
Di seguito riportati gli esempi:





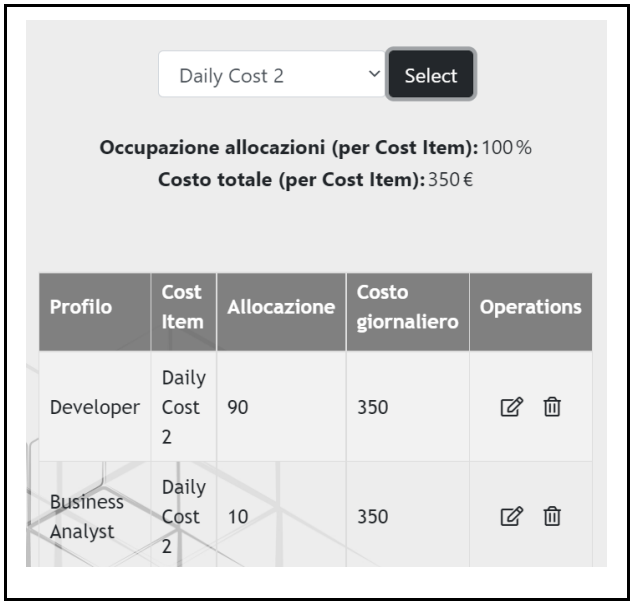


Prendendo in esame la pagina Confskillmix notiamo che rispetto alle altre tabelle che mantengono un certo standard su come vengono mostrate le informazioni, quest'ultima introduce delle differenze.

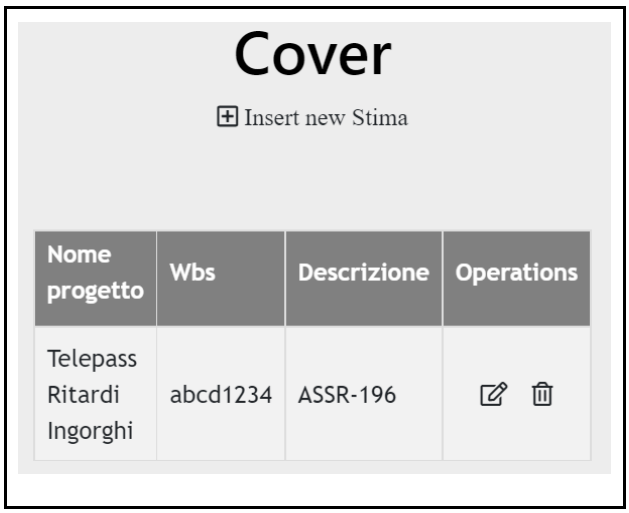


Si noti la presenza di una select: selezionando uno degli elementi (nomi di cost item) che ci vengono mostrati, verrà fatta una ricerca in base a quel nome e il risultato verrà

mostrato nella tabella. Oltretutto, ci verranno mostrati i risultati dell'occupazione delle allocazioni e il costo totale, basati sulla ricerca fatta.

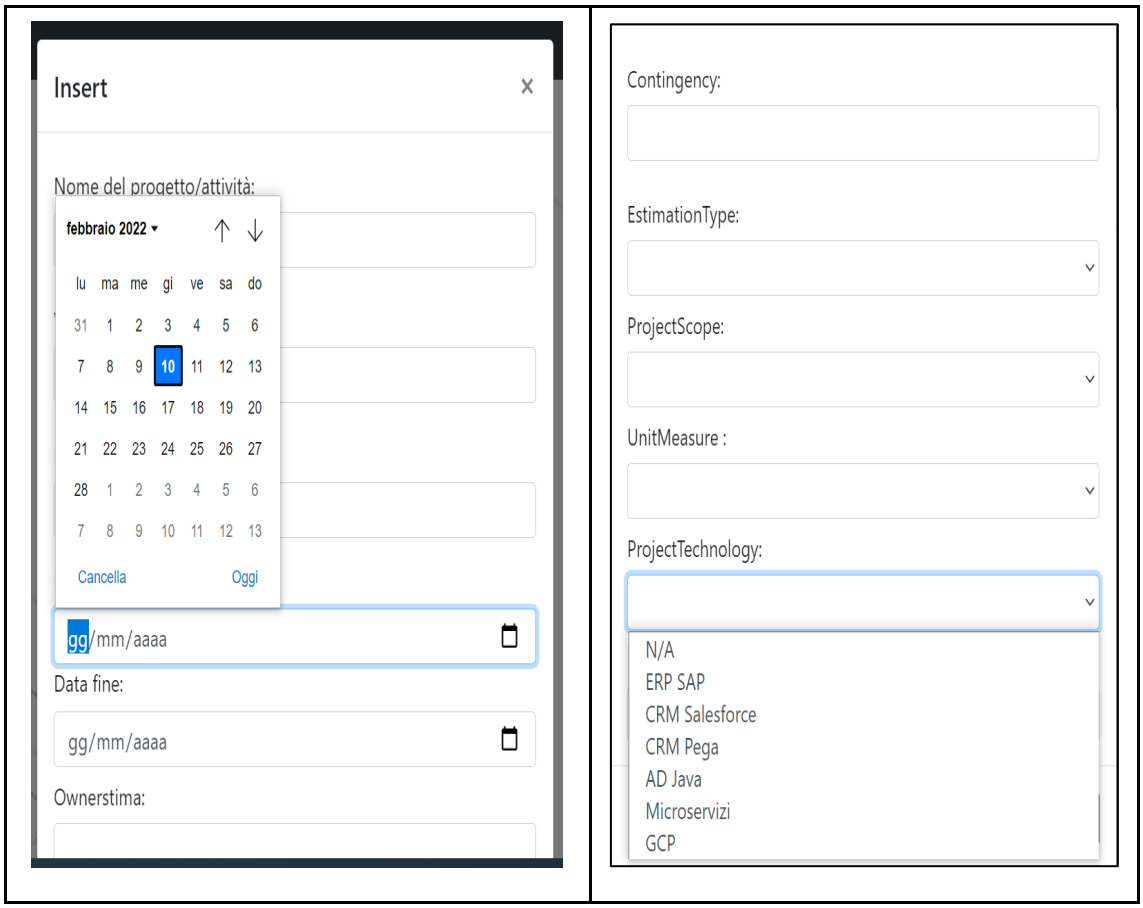


Passiamo alla parte riguardante la stima. La prima pagina che ci viene mostrata è quella della cover. Essa contiene un sunto di quello che sono le informazioni della stima. È possibile inserire ed eliminare elementi, mentre se clicchiamo sul pulsante di edit ci viene mostrata una nuova pagina.

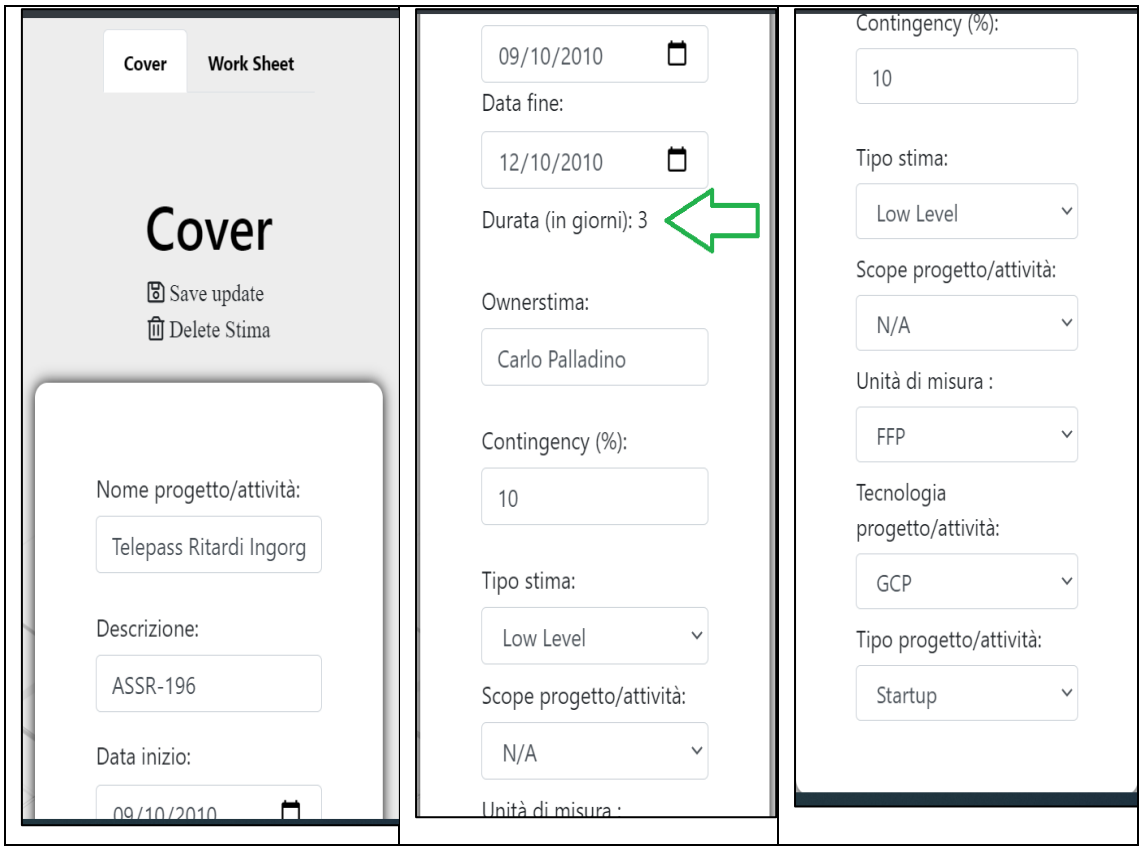


Se clicchiamo sul button che permette l'inserimento, ci verrà, mostrata una modal che basata sull'inserimento di testo o selezione di elementi (ci saranno select per i campi che rappresentano informazioni contenute in altre tabelle).

Questo schema di inserimento è presente in tutte le table che hanno riferimenti di altre tabelle (Confskillmix, Confworkitems, Confworkactivities, Stima, Stimaitem).



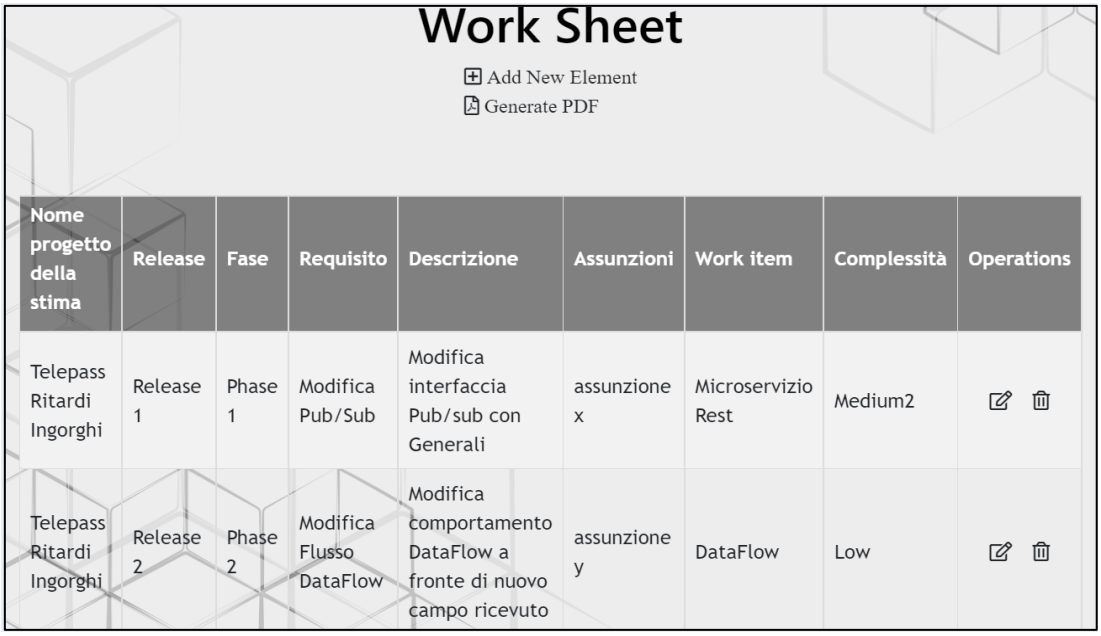
La pagina che ci viene mostrata contiene la Cover della stima con le informazioni complete e il worksheet. Come nel caso della pagina Confarrays, la selezione dell'una o dell'altra tabella è possibile per mezzo di una nav-bar.
La cover ci verrà mostrata in una modal con valori già settati (i valori della stima).
È possibile modificare i parametri, salvando successivamente i cambiamenti, o eliminare la stima.



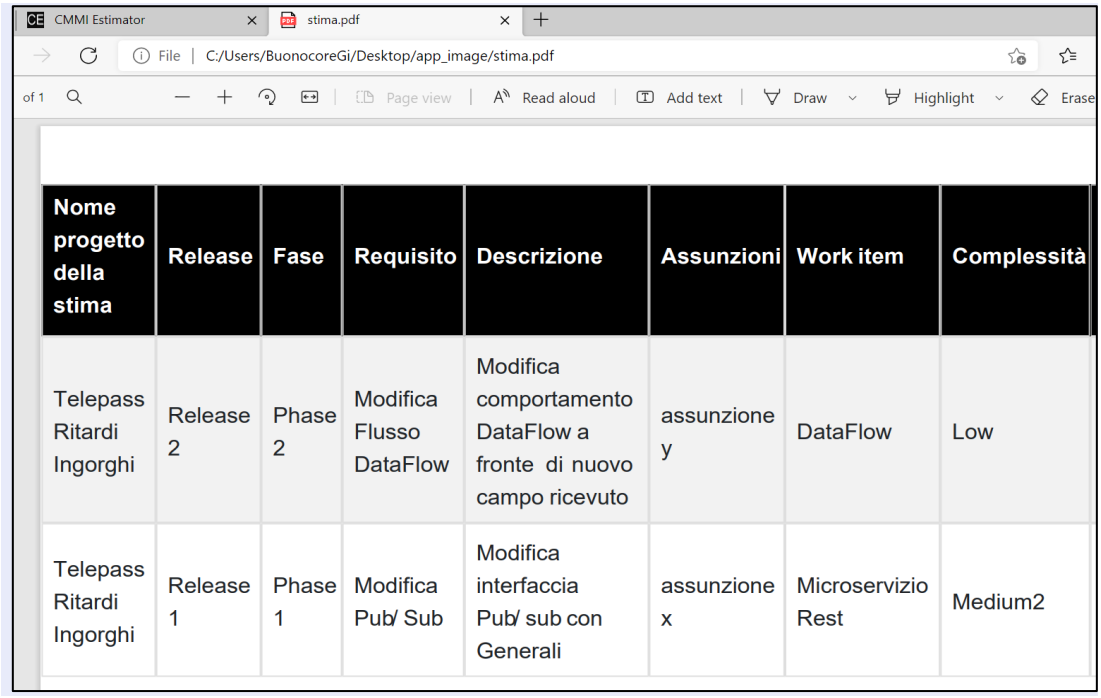
Da notare il dato durata: esso viene calcolato e aggiornato ad ogni aggiornamento di data. Se sulla nav-bar clicchiamo sulla sezione riguardante il Work sheet ci verrà mostrata la seguente tabella. Ricordiamo che per ogni riga della cover è possibile avere più righe nel work sheet.

Come in altre tabelle, anche in questa pagina è possibile aggiungere modificare ed eliminare informazioni dalla tabella.

La differenza di questa pagina, rispetto alle pagine viste finora, è la presenza del tasto che permette di generare e stampare in formato pdf un documento contenente work sheet riguardante la stima presa in esame.



Il documento pdf si presenta così:

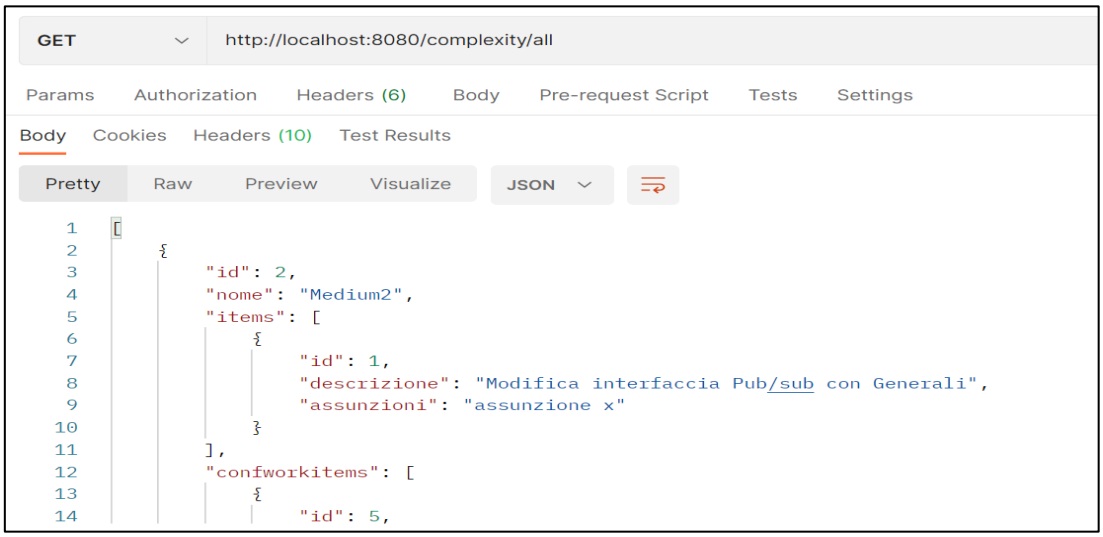


Ad oggi gran parte delle funzionalità del sistema sono implementate. Il prossimo traguardo da compiere in futuro è la creazione di un'app mobile che possa emulare il comportamento di questa applicazione.

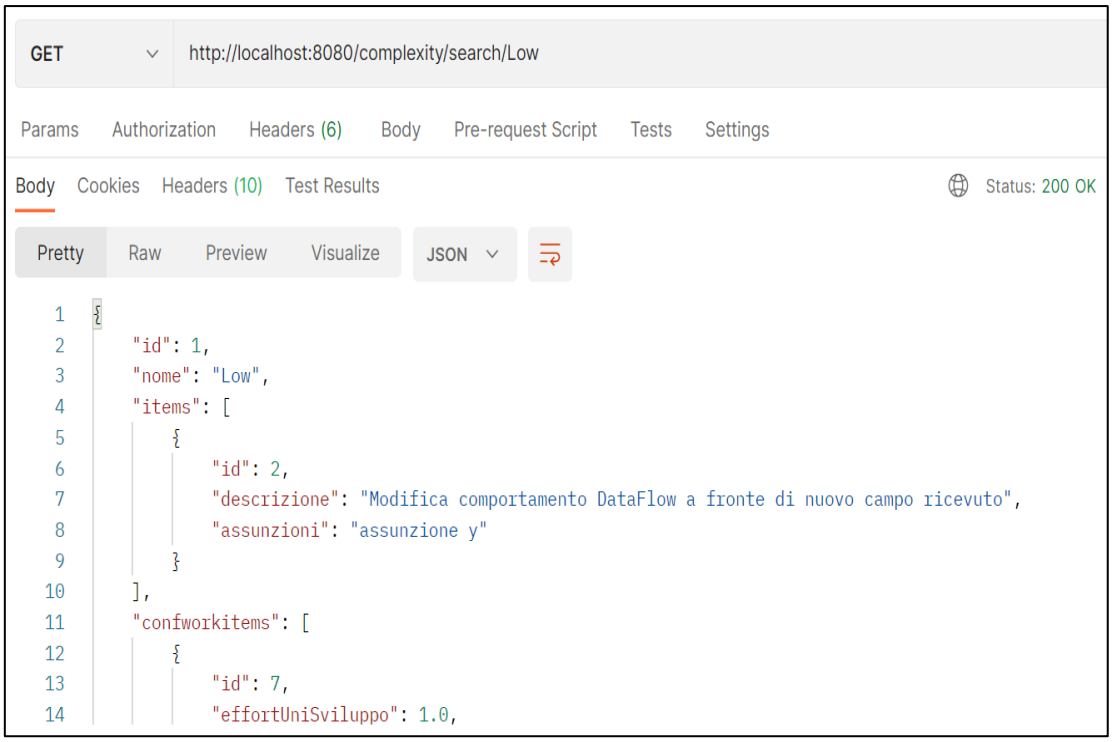
7. Test plan

Per la parte del testing, cioè in questo caso del sollecito dei metodi del back-end, ho deciso di affidarmi ad un tool che si chiama Postman.
Si tratta di uno strumento che permette di eseguire richieste HTTP ad un server di backend. Quando lavoriamo con un altro sviluppatore backend ci permette di condividere le API, ma la sua vera forza è quella di farci sapere tutto di una richiesta HTTP. Possiamo creare la richiesta specificando tutti i parametri possibili, e conoscere ogni informazione della request e della response: headers, body, response ecc.

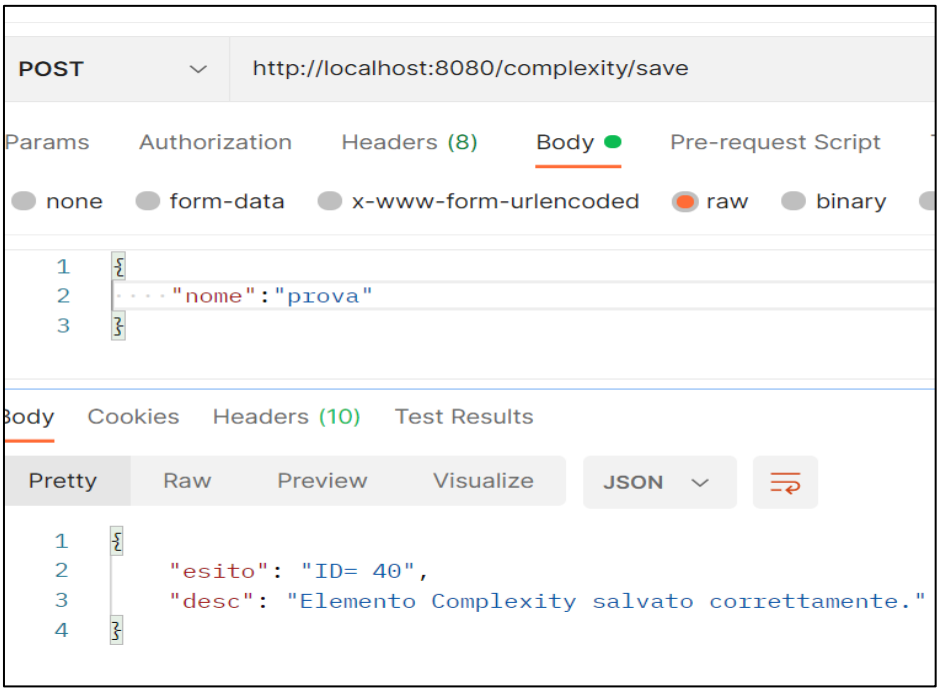
Si sta descrivendo solo una piccola parte delle attività di testing necessarie, cioè quelle legate ai servizi REST forniti dal backend.



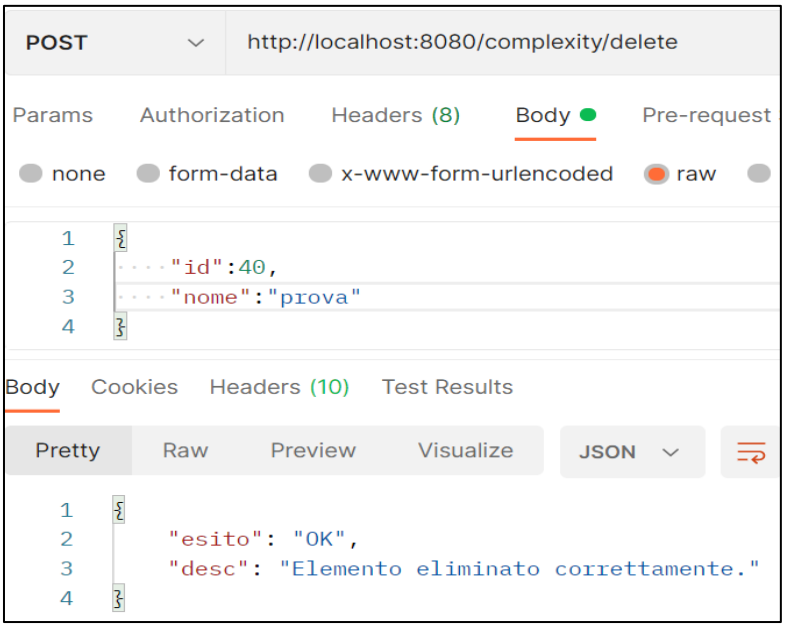
Test ID	1	
Nome	Stampa tabella complexity.	
Descrizione	Ricerca tutti i dati contenuti nei campi della tabella complexity.	
Input	**Risultato atteso**	**Risultato ottenuto**
	Mostrare l'intero contenuto della tabella.	Mostra l'intero contenuto della tabella.



Test ID	2	
Nome	Stampa riga della tabella complexity.	
Descrizione	Ricerca una determinata riga della tabella complexity.	
Input	**Risultato atteso**	**Risultato ottenuto**
Campo nome= ”Low”.	Mostrare la riga corretta della tabella.	Mostra la riga corretta della tabella.



Test ID	3	
Nome	Inserimento riga nella tabella complexity.	
Descrizione	Inserisce una nuova riga nella tabella complexity.	
Input	**Risultato atteso**	**Risultato ottenuto**
Json {“nome”: “prova”}	Inserire correttamente la riga nella tabella.	Inserisce correttamente la riga nella tabella.



Test ID	4	
Nome	Eliminazione riga della tabella complexity.	
Descrizione	Elimina riga nella tabella complexity.	
Input	**Risultato atteso**	**Risultato ottenuto**
Json {“id”:”40”, “nome”: “prova”}	Eliminare correttamente la riga nella tabella.	Elimina correttamente la riga nella tabella.

Nota: Tutte le funzionalità del back-end sono state testate. Il seguente documento è a scopo riassuntivo e dimostrativo.

8. Gestione del progetto

Per la gestione del progetto è stato usato il software Jira.
Jira era stato progettato come strumento per il monitoraggio di bug e ticket, ma oggi si è evoluto in un potente strumento di gestione del lavoro per tutti i casi d'uso, dalla gestione dei requisiti ai test.

Conclusioni sulla gestione del progetto: alla fine del processo lavorativo tutti gli obiettivi pianificati sono stati portati al termine e le tempistiche rispettate.

L’applicazione risulta ottimale per l’utilizzo prefissato.

Conclusioni

In conclusione, si è visto che l'adozione del modello CMMI permette ad un'organizzazione di selezionare una process area o un gruppo di process area e di migliorare i processi che ne fanno parte. Questo tipo di analisi utilizza i livelli di capability per caratterizzare il miglioramento relativo ad una singola process area, offrendo la massima flessibilità e permettendo ad un'organizzazione di scegliere se migliorare le performance di un singolo processo oppure se lavorare su più aree allineate agli obiettivi di business perseguiti.

Si è implementata una web app che definisce uno strumento che rispetti i principi del modello CMMI e che è di aiuto a quelle che sono le esigenze aziendali odierne, specificando studio, architettura e tecnologie per la sua realizzazione.

L'estimator risulta uno strumento efficiente e vincente nel mondo lavorativo, il suo utilizzo può assumere sempre più un'importanza considerevole, con grossi margini di sviluppo e miglioramento.

Bibliografia

I. Ralf Kneuper, *CMMI: Improving Software and Systems Development ProcessesUsing Capability Maturity Model Integration (CMMI-DEV)*, Rocky Nook, 2008

II. Mary Beth Chrissis, Mike Konrad, Sandy Shrum, *CMMI for Development-Guidelines for Process Integration and Product Improvement*, Addison-Wesley Professional, 2011

Sitografia

I. CIO: https://www.cio.com/article/274530/process-improvement-capability-maturity-model-integration-cmmi-definition-and-solutions.html

II. Compliance Power 365: https://www.compliancepower365.com

III. Soft Expert: https://www.softexpert.com/solucao/cmmi/