POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Corso di Laurea Magistrale

in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Sviluppo di un'applicazione client-server per la gestione di progetti software: benefici dell'approccio Domain-driven design

Relatore

prof. Giorgio Bruno

Tutor Aziendale

Andrea Ricossa

Candidato

Marco Iachininoto

Aprile 2018

Alla mia famiglia.

INDICE

1. Introduzione

Oggigiorno le applicazioni web sono sempre più diffuse perché con le moderne tecnologie a disposizione, quali Angular per esempio, è possibile creare applicazioni che offrono un'esperienza d'uso molto simile a quella fornita dalle applicazioni desktop. I moderni browser sono sempre più potenti e permettono di eseguire applicazioni del genere senza problemi.

L'obbiettivo di questo elaborato di tesi è la progettazione, l'analisi e la realizzazione di un'applicazione web per la gestione di progetti software con front-end responsive in Angular e Bootstrap.

L'applicazione ha lo scopo di aiutare ad organizzare il lavoro dei team di sviluppo e di monitorare lo stato di avanzamento dei progetti chiavi in mano. Tramite questa applicazione si possono gestire le anagrafiche delle risorse, delle commesse, dei progetti, si possono associare task ai progetti e si possono anche assegnare task a specifiche risorse. Inoltre, permette anche la consuntivazione giornaliera delle risorse con l'aggiornamento dello stato di avanzamento dei vari task. Infine, offre anche la possibilità di generare dei report sullo stato di avanzamento dei progetti, espresso in giorni rispetto a quelli previsti.

Nel corso del secondo capitolo verrà spiegato nel dettaglio in cosa consiste un'applicazione web e come si sono evolute dalla loro nascita fino ai nostri giorni. Inoltre, verranno descritte le architetture principali che possono essere utilizzate per la loro realizzazione, come l'architettura multi-tier e l'architettura Model-View-Controller. Quindi, verranno introdotte, descritte e confrontate due tra le piattaforme più diffuse utilizzate per lo sviluppo di applicazioni web, cioè la piattaforma .NET e la piattaforma Java.

Il terzo capitolo illustrerà le funzionalità che devono essere sviluppate descrivendo dettagliatamente i requisiti che l'applicazione finale deve soddisfare. Verrà spiegato come deve essere organizzato il menu di navigazione e cosa deve contenere ogni singola pagina associata ad una funzionalità.

Nel corso del quarto capitolo, verranno introdotti i requisiti non funzionali, cioè quelle "regole di buona programmazione" che dovrebbero essere seguite per sviluppare software di qualità. Quindi verranno descritti gli strumenti utilizzati per lo sviluppo dell'applicazione e i vantaggi che offrono nella realizzazione di software di qualità. In particolare, verrà descritto Angular per quanto riguarda il front-end, le Web API 2 Asp.NET per quanto riguarda il back-end, SQL Server per la persistenza dei dati e l'Entity Framework per il mapping object-relational.

Il capitolo finale è dedicato all'implementazione dell'applicazione. In particolare, verrà descritta la struttura sia dell'applicazione Angular per il front-end, sia del web service che espone le API che vengono contattate dal client. Il focus è stato posto sul Domain-driven design: dopo una prima descrizione generale, verrà illustrato come questa tecnica di progettazione software ha influenzato lo sviluppo dell'applicazione oggetto della tesi. In seguito, si passerà alla spiegazione di come sono state implementate le varie funzionalità espresse nei requisiti, mostrando delle schermate di esempio per ognuna delle pagine che costituiscono l'applicazione. Verrà descritto anche il modo in cui si è affrontato il tema

dell’autenticazione e delle autorizzazioni che devono possedere gli utenti dell’applicazione per poter eseguire determinate operazioni.

2. Web Application

Una web application è un'applicazione client-server in cui le funzionalità applicative risiedono su un application server e il client che vi accede esegue in un web browser. Tipicamente quando si parla di web site viene utilizzata la terminologia "web application" se le funzionalità offerte sono simili a quelle di un programma software desktop. Mentre quest'ultimo è sviluppato specificamente per una piattaforma e installato su di essa, le web app vengono utilizzate dal cliente sfruttando l'accesso a Internet o ad una rete locale.

Le applicazioni web portano con sé diversi vantaggi:

- Il client non ha vincoli di compatibilità del sistema operativo per l'accesso a queste applicazioni, ma potrà usufruirne fino a quando il suo browser sarà compatibile.
- Non ci sono problemi di compatibilità sulla versione dell'applicazione poiché gli utenti accedono alla stessa versione.
- Gli utenti possono risparmiare spazio sul proprio hard drive perché non necessitano di installare in locale l'applicazione, ma necessitano solamente di un web browser.
- Si attenuano i problemi legati ai software pirati.

Esempi comuni di web application sono le webmail, le vendite al dettaglio online, i servizi di messagistica istantanea, gli elaboratori di testo online, i fogli di calcolo online e molte altre funzioni.

2.1 Storia delle Web Application

Quando cominciarono a diffondersi le prime applicazioni client-server, a partire dalla metà degli anni Ottanta, l'utente aveva la necessità di installare la parte client dell'applicazione, sotto forma di programma precompilato, direttamente sul proprio personal computer. Ad ogni rilascio di un aggiornamento o di una patch lato server solitamente seguiva anche una modifica dell'applicazione client side e quindi bisognava installare le nuove versioni su tutte le macchine degli utenti. Ciò aveva conseguenze non indifferenti a livello di costi per il produttore. Inoltre, poiché sia la parte client che la parte server dell'applicazione erano legate a uno specifico sistema operativo e ad una particolare architettura, era necessario effettuare il porting verso gli altri sistemi e ciò faceva lievitare ancor di più i prezzi.

D'altra parte, le moderne applicazioni web si basano sull'utilizzo di documenti web che sono scritti in formato standard, come HTML e Javascript. La stragrande maggioranza dei web browser utilizzati dagli utenti è in grado di interpretare nel modo opportuno questi documenti. In particolare, nelle web application il software del client viene scaricato non appena l'utente visita con il web browser la pagina web associata all'applicazione, utilizzando il protocollo HTTP. Gli aggiornamenti del software lato client possono avvenire ogni qualvolta l'utente visiti la pagina web. Il web browser può essere considerato come il client universale per tutte le web application perché riceve le pagine dal server, le interpreta e le visualizza su schermo.

Nelle prime web application il client navigava attraverso una sequenza di pagine, ricevute sotto forma di documenti Web statici. Ed era proprio questa sequenza di pagine visitate a fornire una forma di esperienza interattiva. Ogni volta che la pagina web richiesta veniva modificata, il server veniva chiamato in causa per effettuare il refresh della pagina stessa.

Il 1995 rappresenta una data fondamentale per l'evoluzione delle web application perché Netscape introdusse JavaScript, un linguaggio di scripting client-side. Questo permetteva la realizzazione di pagine web più dinamiche, attraverso l'esecuzione di codice direttamente sul client senza coinvolgere il server. In questo modo era possibile per esempio validare gli input oppure mostrare e nascondere parti della pagina web senza inviare dati al server ma sfruttando solo lo script integrato nella pagina web scaricata.

Nel 1996 Macromedia introdusse Flash, un plug-in che poteva essere incluso nel browser per la realizzazione di animazioni sulle pagine web e per l'arricchimento dell'interfaccia utente, evitando di comunicare con il server. Anche in questo caso si utilizzava un linguaggio di scripting client-side.

Nel 2005 ci fu un'altra svolta fondamentale data dall'introduzione di Ajax, un insieme di tecniche per lo sviluppo web, che permetteva di creare delle web application asincrone e con delle parti client via via più interattive. Questo era possibile utilizzando degli script client-side che contattavano il server per effettuare il download di dati senza dover riscaricare l'intera web page.

Nel 2011 venne ultimato HTML5, tramite il quale si possono realizzare interfacce utente più ricche di animazioni senza il bisogno di utilizzare plug-in client-side (come Flash). HTML5 introdusse nuovi elementi che includono e gestiscono in modo nativo i contenuti multimediali e grafici all'interno delle web page. Nuovi elementi vennero definiti anche per strutturare in modo migliore i documenti. HTML5 portò con sé anche delle API tramite le quali si possono sviluppare web application molto complesse.

2.1.1 Standard per lo sviluppo web odierno e REST

Oggigiorno, HTML5 si è ormai affermato nell'uso tradizionale e fornisce quindi agli sviluppatori web una serie di funzionalità che permettono di spostare nel client molto lavoro che precedentemente era a carico esclusivamente del server. Queste nuove capacità, accompagnate da una maturità via via maggiore delle librerie JavaScript quali JQuery, costituiscono gli standard che bisogna seguire se si vogliono realizzare delle web app competitive.

Contemporaneamente, Representational State Transfer (REST) ha preso il sopravvento nel campo delle architetture per l'interoperabilità delle applicazioni su HTTP. REST descrive un'applicazione in termini di risorse e di operazioni standard. Le risorse sono il concetto fondamentale di questa architettura, in cui ogni risorsa è associata ad un identificatore globale (URI) e rappresenta le entità del mondo reale. Le operazioni standard, che vengono mappate sui metodi HTTP (GET, POST, PUT, DELETE), rappresentano le operazioni disponibili su queste risorse. Per esempio, per poter leggere le informazioni associate ad un Customer, identificato con un certo ID, si utilizza il metodo GET su un URI del tipo *http://www.esempio.com/Customers/20*.

Al giorno d'oggi, le applicazioni web non offrono solamente documenti HTML, ma devono offrire anche dati in formato JSON o XML destinati alle più disparate tecnologie client, quali AJAX o le applicazioni native per gli smartphone. REST permette di ottenere ciò in modo semplice, eliminando la storica distinzione tra servizi web e applicazioni web.

2.2 Architettura multi-tier

È solito suddividere le applicazioni in livelli logici (detti tier) ed assegnare ad ognuno di essi un compito differente. Le web application, però, non possono essere viste allo stesso modo in cui vengono viste le applicazioni tradizionali. Difatti queste ultime sono costituite da un unico tier, contenuto all'interno della macchina che agisce da client. Di contro, le web app per loro natura si riconducono ad una stratificazione su più livelli.

Con il termine architettura multi-tier (detta anche n-tier architecture) si intende un'architettura software in cui da un punto di vista logico le funzionalità del software sono separate su strati diversi che comunicano tra di loro. Ognuno di questi strati può comunicare solamente con gli strati ad esso adiacenti, come se ogni strato agisse da client verso uno dei due strati adiacenti e da server verso l'altro strato adiacente (ad eccezione per i due strati estremi che possono comportarsi solo da client o da server). Tali tier possono essere contenuti sia all'interno di un'unica macchina, sia su macchine diverse. La prima a diffondersi negli anni Novanta è stata l'architettura a due livelli, per poi passare ad un'architettura a tre livelli negli anni 2000. Oggi non è raro trovare architetture che ospitano più di tre livelli e per questo si utilizza molto anche il termine architettura multi-tier.

I vantaggi di suddividere l'architettura delle applicazioni in più livelli sono la flessibilità e la scalabilità. Infatti, nel momento in cui si vuole modificare una funzionalità di un'applicazione o aggiungerne una nuova, si agisce solo nel livello al quale appartiene la funzionalità in esame senza dover riscrivere l'intera applicazione. La conseguenza principale è che in questo modo le applicazioni sono più mantenibili.

2.2.1 Architettura three-tier

Con il termine architettura three-tier si intende un'architettura client-server in cui l'interfaccia utente, la logica dell'applicazione e l'archiviazione dei dati con il relativo accesso agli stessi sono sviluppati e mantenuti sotto forma di moduli indipendenti. Nel caso in cui si presentano dei cambiamenti nei requisiti o nella tecnologia, un'applicazione modulare consente di aggiornare o sostituire qualsiasi dei tre livelli senza dover modificare gli altri.

Tipicamente l'interfaccia utente è eseguita sul pc dell'utente che utilizza l'applicazione, la logica risiede in uno o più moduli diversi che costituiscono l'applicazione server e l'archiviazione dei dati è gestita da un RDBMS che esegue in un database server. È possibile organizzare il livello intermedio in più tier e in questo caso è più consono utilizzare il termine architettura multi-tier.

I livelli che costituiscono l'architettura three-tier sono:

- Livello di presentazione:

Questo è il livello più alto e si può associare al motore di rendering del web browser, che visualizza le informazioni relative ai servizi dell'applicazione (front-end), come per esempio l'elenco delle e-mail ricevute in una webmail. L'interfaccia utente è costituita dall'insieme di web page scritte in HTML, con l'aggiunta di CSS per arricchirne gli stili e Javascript, o altri framework, per renderle più dinamiche. Quindi l'interfaccia utente sostituisce le GUI tipiche dei programmi software desktop.

- Livello di applicazione (detto anche logica di business, logica di primo livello o livello intermedio):
  Questo livello è associato al motore applicativo (back-end) che risiede nell'application server ed è costituito dal codice sorgente scritto utilizzando una qualche tecnologia per lo sviluppo di contenuti web dinamici (come Asp.NET, PHP, JSP/Java, Ruby on Rails). Quindi questo livello gestisce le funzionalità dell'applicazione eseguendo delle elaborazioni dettagliate in modo da soddisfare le richieste del client. È questo il tier che rende diversa una web app nei confronti di un normale sito web.
- Livello dei dati:
  Questo livello è associato al server che agisce da database. In realtà non è detto che tale server database risieda in una macchina diversa da quella che contiene l'application server, perché entrambi potrebbero essere ospitati nella stessa macchina. Vari motori di database possono essere utilizzati per gestire la persistenza dei dati, come MySQL, MSSql, Oracle, DB2 o molti altri ancora. Allo stesso tempo sono vari i tool che possono essere utilizzati per interrogare il database, come Hibernate o Ibatis. Le richieste di accesso al database, siano esse in lettura o in scrittura, sono ricevute dal livello di applicazione.

Quanto è stato detto riguardo i livelli che costituiscono un'architettura three-tier è mostrato nella "Figura 1".

Ricapitolando, il web browser del client utilizza il protocollo HTTP/HTTPS per inviare richieste al back-end. Questo interpreta ciò che è contenuto nella richiesta, esegue eventuali elaborazioni e a sua volta genera altre richieste che vengono destinate al motore del database. Non appena il livello intermedio riceve i dati dal database, genera il risultato in un output che sarà interpretato dal browser e restituito all'utente nella forma di una pagina web.

Per le applicazioni più complesse una soluzione a 3 livelli potrebbe non bastare e quindi può essere conveniente passare ad un approccio n-tier. Il vantaggio principale è che in questo modo la logica di business viene suddivisa in un modello più granulare. Si può pensare anche di introdurre un livello di integrazione al di sopra del livello dei dati che fornisce delle API al livello di applicazione. Queste API espongono l'insieme dei metodi che possono essere utilizzati per gestire i dati salvati senza creare dipendenze con lo specifico motore di database utilizzato nell'applicazione. In questo modo si può passare da un motore ad un altro senza che il livello di applicazione se ne renda conto.

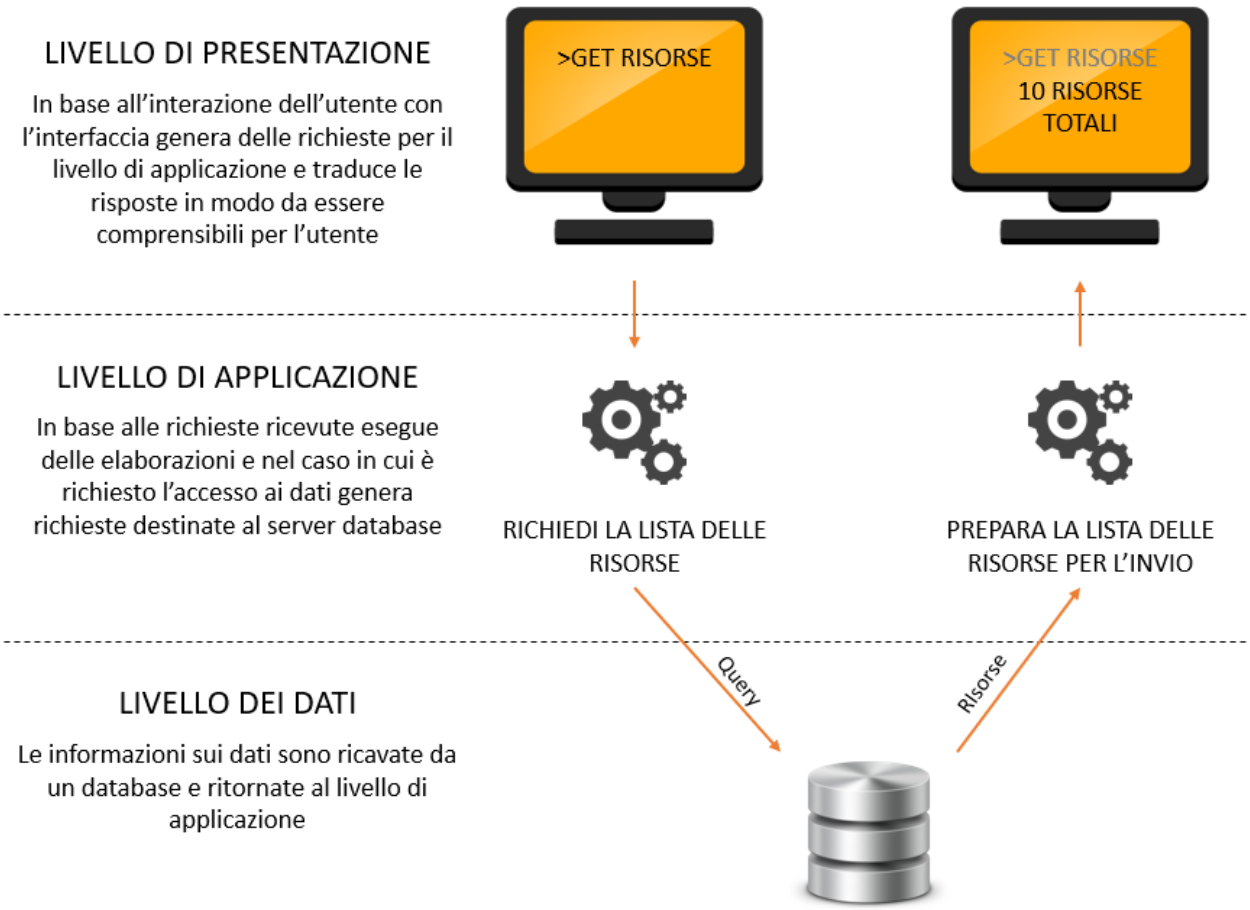


***Figura 1**: Rappresentazione grafica di un'applicazione con architettura three-tier*

2.3 Architettura Model-View-Controller

Il Model-View-Controller è un pattern architetturale che divide la logica di presentazione dei dati dalla logica di business ed è diffusissimo nello sviluppo software, soprattutto se parliamo di programmazione orientata agli oggetti. L'MVC permette quindi di separare il modo in cui l'informazione è rappresentata dal modo in cui è presentata e accettata dall'utente. Il disaccoppiamento delle tre componenti che costituiscono il pattern favorisce il riuso del codice: gli sviluppatori possono riutilizzare in modo semplice e veloce uno dei componenti in un'altra applicazione. Per esempio, uno stesso view, o due view simili, possono essere utilizzate per due applicazioni che gestiscono dei dati diversi, visto che il view (come spiegato nel paragrafo successivo) stabilisce solamente come i dati devono essere mostrati all'utente. Inoltre, la creazione di componenti indipendenti uno dall'altro favorisce uno sviluppo parallelo: gli sviluppatori possono lavorare sui diversi componenti parallelamente senza bloccarsi a vicenda. Per esempio, gli sviluppatori back-end possono strutturare i dati e il modo in cui l'utente interagisce con essi senza la necessità che la user interface sia completa. Allo stesso modo, gli sviluppatori front-end possono progettare il layout senza il bisogno di avere pronta la struttura relativa ai dati. Ultimo aspetto, ma non meno importante, è che la separazione delle responsabilità semplifica eventuali future modifiche o sviluppi del codice.

Questa architettura è così utilizzata nella progettazione delle web application, e non solo, che la maggior parte dei linguaggi di programmazione come Java, C#, Ruby o PHP integrano dei framework MVC che possono essere sfruttati direttamente out-of-the-box.

2.3.1 Storia

Il termine model-view-controller è stato introdotto alla fine degli anni Settanta nell'ambito del progetto Smalltalk allo Xerox PARC, dove era stato utilizzato per indicare un modo di organizzare alcune delle prime applicazioni dotate di interfaccia grafica. Solo nel decennio successivo il pattern MVC cominciò ad essere visto come un concetto generale. Infatti, alcuni dei dettagli più particolari del pattern originale erano associati a concetti specifici di Smalltalk, come per esempio screen e tool, ma i concetti generali possono essere associati alle generiche applicazioni, soprattutto alle web application.

Successivamente questa architettura si evolse dando vita a delle varianti, quali hierarchical model-view-controller (HMVC), model-view-adapter (MVA), model-view-presenter (MVP) e model-view-viewmodel (MVVM).

L'utilizzo del pattern MVC assunse sempre più importanza nell'ambito delle applicazioni web dal momento in cui nel 1996 venne introdotto WebObjects, un framework per le applicazioni web server-based sviluppato da NeXT, che era scritto in linguaggio Objective-C. Non appena venne effettuata la portabilità di WebObjects su Java, il pattern MVC divenne molto popolare anche tra gli sviluppatori Java. I framework seguenti che si basavano su Java, come per esempio Spring (rilasciato nel 2002), non fecero altro che rafforzare ancor di più il legame tra Java e il pattern MVC. Successivamente l'introduzione nel 2005 di framework come Django (per Python) e Rails (per Ruby) e quella a fine anni 2000 del framework Asp.NET MVC diedero popolarità all'architettura MVC anche al di fuori dell'ambiente in cui era nato ed era stato a lungo popolare.

2.3.2 Componenti

I componenti software che vengono distinti nel pattern MVC sono tre:

- Il model:
  È il componente principale del pattern e serve a determinare il dominio dell'applicazione, senza considerare l'interfaccia utente. Per esempio, in un'applicazione bancaria il model deve includere tutto ciò che facente parte del mondo bancario viene supportato all'interno dell'applicazione stessa, come i clienti, i conti correnti, il credito ad essi associato, le operazioni che permettono di fare dei versamenti o dei prelievi. Il model si deve occupare anche di mantenere i dati in uno stato consistente, assicurandosi che un cliente non possa prelevare più di quanto disponibile sul suo conto o più denaro liquido di quello che al momento ha a disposizione la banca. Il model incapsula lo stato dell'app e si occupa dell'accesso ai dati utilizzati all'interno dell'applicazione fornendo un opportuno set di metodi. Per implementare nel modo corretto il model conviene seguire quanto più possibile le tecniche di progettazione object oriented in modo da astrarre al meglio i concetti importati dal mondo reale.
- Il view:

È il componente che visualizza i dati contenuti nel model e gestisce l'interazione con l'utente costruendo per esso un'interfaccia grafica (GUI). Ogni rappresentazione in output di informazioni, come per esempio delle tabelle o dei grafici, costituisce una vista. Le stesse informazioni potrebbero essere mostrate con viste differenti, come per esempio un diagramma per la gestione e una tabella per l'amministrazione. Per far sì che i dati visualizzati siano aggiornati è possibile seguire la strategia "push model" o la strategia "pull model". La prima segue il pattern Observer, il che vuol dire che i view si registrano come osservatori del model. In questo modo i view colgono eventuali aggiornamenti dei dati in tempo reale poiché ricevono delle notifiche direttamente dal model. La seconda strategia si basa sul fatto che il view richiede gli aggiornamenti quando lo ritiene opportuno.

- Il controller:
  È il componente che riceve dal view i comandi in input che sono stati generati dall'utente, quindi sulla base di questi comandi esegue delle operazioni che possono modificare lo stato del model e sceglie il view appropriato da mostrare all'utente.

Ricapitolando, ogni pezzo dell'architettura MVC è ben definito ad autosufficiente, esaltando il principio della *separazione degli interessi* (*separation of concerns*): la logica che manipola i dati del model si trova solo all'interno del model, quella che visualizza i dati solo all'interno dei view e quella che gestisce gli input degli utenti solo all'interno dei controller. Volendosi ricollegare ai tre livelli di un'architettura three-tier, il controller e il model implementano la logica di business dell'applicazione, mentre il view implementa l'interfaccia utente.

2.3.3 Interazioni tra i componenti

L'architettura MVC definisce anche come devono interagire i tre componenti software. Il model risponde alle domande sullo stato che provengono dal view e riceve dal controller i comandi su come deve essere aggiornato lo stato dell'applicazione. Il view richiede lo stato dell'applicazione al model e quindi presenta i dati all'utente in un certo formato, dopo essere stato triggerato dal controller. Inoltre, per permettere l'interazione con l'utente il view setta i listener per l'input. Infine, il controller riceve gli input dell'utente, li valida se è prevista una forma di validazione, li mappa agli aggiornamenti del model e seleziona il view da mostrare al seguito dell'ultima interazione con l'utente. La "Figura 2" mostra il diagramma delle interazioni tra i componenti del pattern MVC.

Quanto detto ci permette di capire perché questo pattern ha raggiunto un enorme popolarità al giorno d'oggi nell'ambito dello sviluppo delle web application. Infatti, l'interazione dell'utente con un'applicazione MVC segue un ciclo naturale: l'utente decide di effettuare un'azione e di conseguenza l'applicazione modifica il modello dei dati e fornisce all'utente un view aggiornato. Questo ciclo si ripete fintantoché l'utente continua ad usare l'applicazione e si adatta perfettamente alle applicazioni web che si basano su una serie di richieste e risposte HTTP.

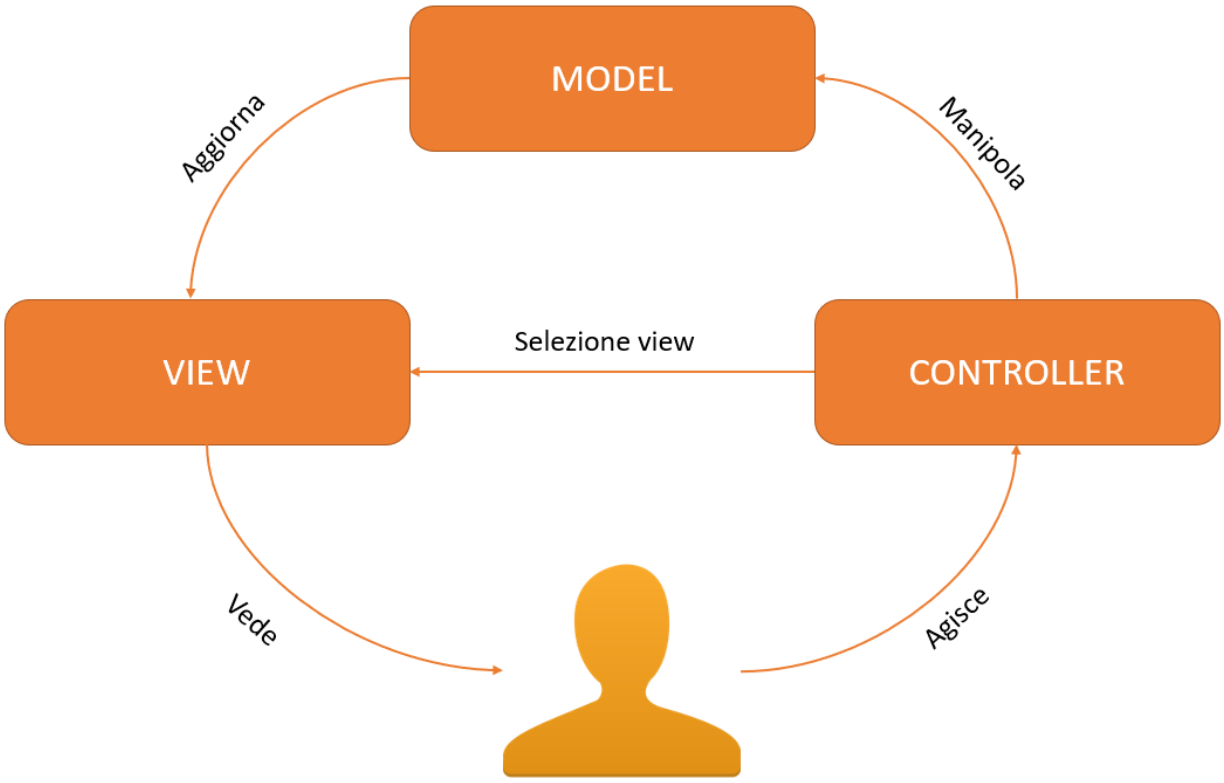


***Figura 2**: Diagramma delle interazioni nel pattern MVC*

2.3.4 Utilizzo nelle web application

Sin dalla sua comparsa l'architettura MVC è stata implementata lato server. In tempi più recenti, grazie alla diffusione di framework sempre più potenti per lo sviluppo client-side, basati per esempio su JavaScript, sono comparse le prime implementazioni lato client.

Per quanto riguarda il lato server, sono molte le tecnologie moderne che includono in modo implicito o in modo esplicito il pattern. Tra queste tecnologie possiamo elencare framework basati su PHP (come CodeIgniter), su Ruby (come Ruby on Rails), su Python (come Django), su java (come Spring e JSF) e su .NET (come Asp.NET MVC). In un approccio del genere il client invia richieste di hyperlink o sottomissioni di form e riceve dal view una pagina web completa e aggiornata.

D'altro canto, negli ultimi anni sono stati creati i primi framework che implementano il pattern interamente in JavaScript poiché, grazie alla diffusione di AJAX, è cresciuta la richiesta di applicazioni web che facciano chiamate asincrone al server senza effettuare il redirect alla ricezione delle risposte. Tra questi framework possiamo annoverare Backbone, React, AngularJS e molti altri.

2.4 Piattaforme utilizzabili

Le piattaforme software principali che possono essere utilizzate per lo sviluppo di applicazioni web sono due: la piattaforma .NET e la piattaforma Java (Java Platform).

2.4.1 .NET

.NET è una piattaforma sviluppata da Microsoft principalmente per sistemi operativi Windows. All'interno di questa piattaforma agisce una grossa libreria di classi che prende il nome di Framework Class Library (FCL), di cui una delle caratteristiche principali è che ognuno dei linguaggi messi a disposizione può utilizzare il codice scritto in uno degli altri linguaggi che appartengono a .NET. Le applicazioni scritte per questa piattaforma vengono eseguite nel contesto di una virtual machine, il Common Language Runtime (CLR), che offre meccanismi per la gestione della sicurezza, della memoria e delle eccezioni. Il codice scritto in .NET prende il nome di "codice gestito" proprio per questo motivo. FCL e CLR assieme rappresentano il cuore della piattaforma .NET.

All'interno della libreria FCL sono presenti classi per la gestione della user interface, dell'accesso ai dati, della connessione col database, dello sviluppo di applicazioni web e molto altro. Quindi il codice scritto dal programmatore si va ad aggiungere a quello già offerto dalla piattaforma e eventualmente a quello fornito da altre librerie incluse dallo sviluppatore stesso.

La piattaforma .NET è accompagnata da una famiglia di piattaforme destinate per esempio alla programmazione mobile, ai sistemi embedded o a sistemi operativi alternativi.

2.4.1.1 Common Language Infrastructure

L'architettura della piattaforma è costituita da diversi elementi. Il primo di questi è la Common Language Infrastructure (CLI) che fornisce un contesto indipendente dal linguaggio utilizzato per lo sviluppo dell'applicazione, comprendente delle funzioni per la gestione del garbage collector, della sicurezza e della gestione delle eccezioni. Questo vuol dire che queste funzioni sono disponibili per tutti i linguaggi che sono supportati dalla piattaforma .NET. Il Common Language Runtime, che implementa la CLI, rappresenta il motore di esecuzione della piattaforma .NET. Quindi tutte le applicazioni sono eseguite sotto il controllo del CLR.

Il codice delle app potrebbe essere scritto in uno qualsiasi dei linguaggi .NET (come C#, VB o J#), ma qualsiasi sia il linguaggio utilizzato il codice viene compilato in un linguaggio indipendente dalla piattaforma hardware che prende il nome di CIL (Common Intermediate Language). Al momento dell'esecuzione il CLR utilizza un compilatore just-in-time (JIT) che trasforma il CIL in un codice macchina compatibile con la piattaforma hardware utilizzata. Questo vuol dire che a partire dallo stesso CIL si possono utilizzare diversi compilatori JIT per produrre codici in linguaggio macchina destinati a processori diversi.

Il codice eseguibile generato dal compilatore just-in-time viene salvato in quella che prende il nome di .NET Native Image Cache. Ciò permette di velocizzare i lanci dell'applicazione successivi al primo, che invece è solitamente più lento. Per rendere più veloce anche il primo lancio Microsoft ha messo a disposizione degli sviluppatori l'utility Native Image Generator, che permette di effettuare manualmente la compilazione ahead-of-time (AOT).

2.4.1.2 Assembly CLI

Il codice CIL è salvato nelle assembly CLI con il formato Portable Executable (PE), utilizzato anche da tutte le DLL e dai file eseguibili EXE. Ogni assembly è formato da uno o più file: uno

di questi file deve fungere da manifesto e contenere i metadati per l'assembly. Il nome completo di un assembly è dato dall'unione del suo nome testuale, dal numero di versione e dalla chiave pubblica associata alla chiave privata con cui l'assembly è stato firmato.

2.4.1.3 Librerie di classi con particolare attenzione a LINQ

Le librerie delle classi offerte dalla piattaforma sono organizzate seguendo una gerarchia di namespace e le due radici principali sono *System.** e *Microsoft.**. Tramite queste classi è possibile effettuare svariate operazioni, quali la lettura e la scrittura di file, l'accesso al database, la manipolazione di documenti XML. Queste librerie sono accessibili da tutti i linguaggi CLI e sono suddivise in due parti: la Base Class Library (BCL) e la Framework Class Library (FCL). Nella prima rientra solo un piccolo sottoinsieme di classi che costituiscono le API di base del CLR. L'insieme delle classi restanti fanno parte del secondo gruppo, come quelle relative all'Asp.NET, al Windows Presentation Foundation (WPF), all'ADO.NET, al Language Integrated Query (LINQ). Per l'accesso alle librerie di terze parti è possibile utilizzare il package manager chiamato NuGet.

Al di sopra delle librerie di classi è possibile creare applicazioni avvalendosi di uno tra i diversi modelli di applicazioni offerte dalla piattaforma, come applicazioni Console, Windows Form, WPF, Asp.NET e Asp.NET core.

LINQ merita un'attenzione particolare perché è un potente linguaggio per effettuare query, disponibile sia in C# che in VB.NET. Offre un modo unico per accedere a dati che possono provenire da diversi tipi di sorgenti, sia in lettura che in scrittura. La "Figura 3" permette di avere un quadro generale su come può essere utilizzato LINQ.

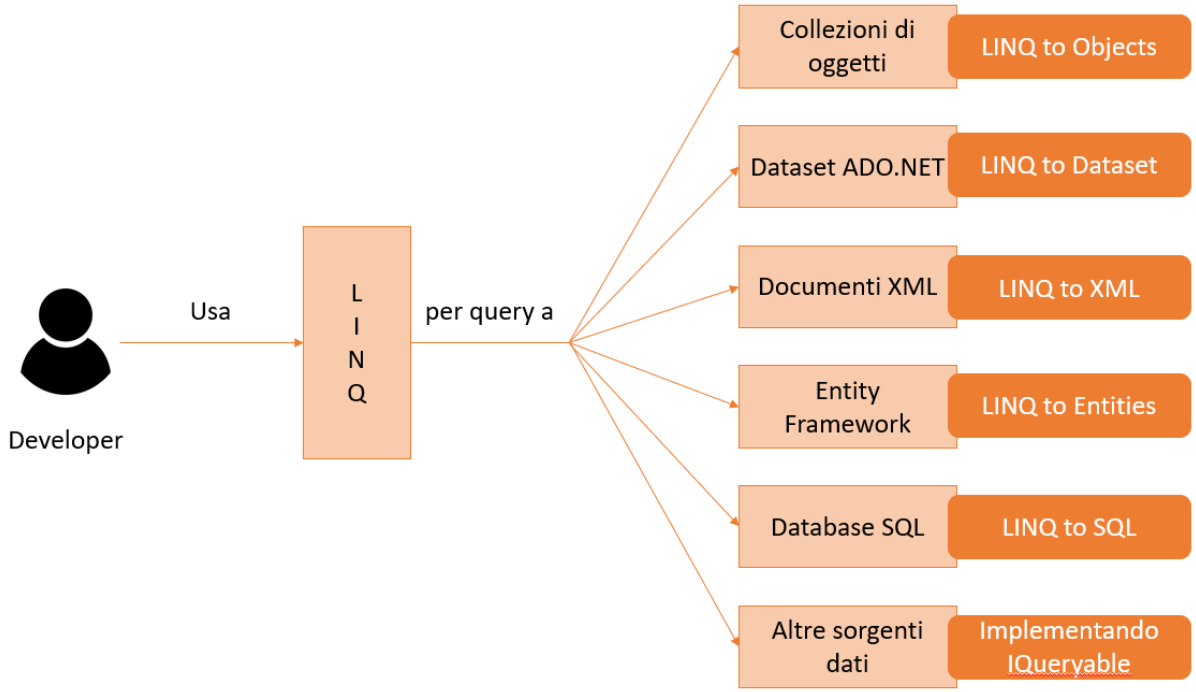


***Figura 3**: Utilizzo di LINQ*

LINQ introduce diversi vantaggi:

- Gli sviluppatori non hanno bisogno di imparare un linguaggio di query diverso in base al tipo di dato al quale vogliono accedere, ma utilizzano sempre la stessa sintassi per ogni sorgente di dati.
- Riduce la quantità di codice scritto.
- Rende il codice più leggibile.
- La presenza di eventuali errori sui tipi di dato presenti nelle query viene controllata durante la fase di compilazione.

LINQ è un insieme di metodi di estensione per le classi che implementano le interfacce *IEnumerable* e *IQueryable*. Rientrano nel primo caso tutti i tipi di collezioni inclusi nel namespace System.Collection.Generic, come *List<T>, SortedList<T>, Queue<T>, LinkedList<T>*. Invece, rientrano nel secondo caso per esempio le API offerte dall'Entity Framework (che verrà descritto approfonditamente nel capitolo 4) per l'accesso ai dati contenuti in un database sottostante come SQL Server. La differenza principale tra le due interfacce è che nel primo caso una query viene effettuata su una collezione di oggetti che risiedono tutti in memoria, mentre nel secondo caso i filtri introdotti in una query vengono applicati direttamente nella sorgente di dati remota, come per esempio un server database.

LINQ offre due sintassi alternative nella scrittura di query: la query syntax e la method syntax. La query syntax è molto simile alla sintassi del linguaggio SQL. Un esempio è il seguente:

```
// collection di studenti
IList<Studente> listaStudenti = new List<Studente>() {
        new Studente() { IDStudente = 1, Nome = "Marco", Eta = 13} ,
        new Studente() { IDStudente = 2, Nome = "Antonio",  Eta = 21 } ,
        new Studente() { IDStudente = 3, Nome = "Giovanni",  Eta = 18 } ,
        new Studente() { IDStudente = 4, Nome = "Luca" , Eta = 20} ,
        new Studente() { IDStudente = 5, Nome = "Matteo" , Eta = 15 }
    };

// LINQ Query Syntax per trovare gli studenti giovani
var studentiGiovani = from s in listaStudenti
                      where s. Eta > 12 && s. Eta < 20
                      select s;
```

La method syntax, d'altro canto, fa un uno esplicito dei metodi di estensione definiti per le interfacce *IEnumerable* e *IQueryable* e delle funzioni lambda. Un esempio è il seguente:

```
// collection di studenti
IList<Studente> listaStudenti = new List<Studente>() {
        new Studente() { IDStudente = 1, Nome = "Marco", Eta = 13} ,
        new Studente() { IDStudente = 2, Nome = "Antonio",  Eta = 21 } ,
        new Studente() { IDStudente = 3, Nome = "Giovanni",  Eta = 18 } ,
        new Studente() { IDStudente = 4, Nome = "Luca" , Eta = 20} ,
        new Studente() { IDStudente = 5, Nome = "Matteo" , Eta = 15 }
    };

// LINQ Method Syntax per trovare gli studenti giovani
var studentiGiovani = listaStudenti.Where(s => s.Eta > 12 && s.Eta < 20)
                                   .ToList<Studente>();
```

2.4.1.4 Indipendenza dal linguaggio e sicurezza dei tipi

Tutti i possibili tipi di dato e i costrutti di programmazione supportati dal CLR assieme alle loro possibili interazioni sono definite all'interno del Common Type System (CTS). Questo permette alle librerie e alle applicazioni scritte utilizzando uno dei linguaggi .NET di scambiarsi tipi e istanze di oggetti.

Sia il CTS che il CLR si occupano di assicurare la sicurezza dei tipi non permettendo cast mal definiti, invocazioni di metodi errate ed eventuali problemi di memoria durante gli accessi ad un oggetto.

2.4.1.5 Portabilità

Per quanto riguarda la portabilità, Microsoft ha implementato la piattaforma solo sui propri sistemi Windows. Il framework è stato però progettato in modo da poter essere sviluppato anche su altre piattaforme. Infatti, Microsoft ha sottomesso le specifiche del CLI (CTS, CIL, BCL e FCL) e C# sia all'ECMA che all'ISO. In questo modo il framework e i suoi linguaggi possono essere implementati anche su altre piattaforme.

2.4.1.6 Sicurezza

Il framework .NET utilizza il Code Access Security (CAS) per evitare che del codice non fidato possa effettuare delle azioni privilegiate. Nel momento in cui il CLR deve effettuare il caricamento di un assembly otterrà quella che prende il nome di evidenza, che è associata allo specifico assembly. Tramite questa evidenza, il CLR associa l'assembly ad un *code group* che a sua volta, rappresenta un insieme di permessi. Il codice che necessita di eseguire delle azioni con dei privilegi effettua un'opportuna richiesta. A questo punto, il CLR ripercorre lo stack delle chiamate per verificare l'insieme dei permessi associati ad ogni metodo nello stack: se anche uno solo degli assembly non possiede il permesso richiesto allora viene sollevata un'eccezione di sicurezza.

Sempre restando in tema di sicurezza, il bytecode del CIL se non è offuscato può essere reverse-engineered in modo semplice. All'inizio degli anni 2000 Microsoft ha introdotto Dotfuscator, per evitare che venga generato del codice intermedio non offuscato.

2.4.1.7 Gestione della memoria

Per quanto riguarda la gestione della memoria, lo sviluppatore non si deve preoccupare né di allocare la memoria né di rilasciarla. Il CLR si fa carico di questo lavoro e capisce autonomamente quando la memoria può essere rilasciata in sicurezza. Gli oggetti che vengono istanziati in .NET ricevono una porzione di memoria da quello che prende il nome di *managed heap*, che è gestito dal CLR. Fintantoché esiste un riferimento associato ad un oggetto, allora quest'ultimo non può essere eleggibile per la cancellazione; dal momento in cui l'oggetto non è più accessibile perché nessun riferimento punta ad esso, allora la memoria allocata ad esso potrà essere deallocata dal garbage collector (GC).

Il garbage collector viene eseguito periodicamente su un thread diverso da quello dell'applicazione e controlla l'eventuale presenza di oggetti non più accessibili in modo da liberare la memoria che era stata allocata per essi. Il GC è non deterministico perché entra in esecuzione solo quando la quantità di memoria utilizzata ha superato una certa soglia o quando il sistema ha bisogno di memoria libera. Ogni applicazione ha un insieme di root, che sono dei puntatori agli oggetti del managed heap. Tra questi ci sono i riferimenti agli oggetti statici e a quelli definiti come variabili locali o parametri dei metodi nello scope corrente. Nel momento in cui il GC entra in esecuzione l'applicazione viene messa in pausa e il garbage collector contrassegna come raggiungibili tutti gli oggetti puntati dall'insieme di root ed eventuali altri oggetti in essi incapsulati. Poiché l'heap era stato precedentemente allocato in modo contiguo, tutti gli oggetti che non sono stati contrassegnati diventano garbage, il che vuol dire che la memoria che era stata allocata per essi è diventata spazio libero. Questa operazione prende il nome di *mark*. A questo punto, dato che ci potrebbero essere blocchi di memoria libera tra gli oggetti che erano stati allocati inizialmente, viene eseguita un'operazione di *sweep* che serve a compattare la memoria. Al termine dell'esecuzione del GC l'applicazione può riprendere l'esecuzione, anche se nelle ultime versioni della piattaforma .NET queste pause sono impercettibili perché il garbage collector è eseguito in background.

Un'altra caratteristica del garbage collector è che esso è generazionale perché ad ogni oggetto viene assegnata una generazione. Gli oggetti creati per ultimi rappresentano la Generazione 0; quelli che resistono ad un'operazione di garbage collection diventano Generazione 1; gli oggetti di Generazione 1 che sopravvivono ad un'altra garbage collection sono contrassegnati come Generazione 2. Avere un elevato valore di generazione vuol dire essere soggetti a meno operazioni di garbage collection. Questo permette di migliorare l'efficienza del GC perché un numero minore di oggetti necessità di essere compattato.

2.4.2 Java

Java è una piattaforma software sviluppata originariamente da Sun Microsystem, che è stata poi acquisita da Oracle. Con l'utilizzo di Java è possibile sviluppare applicazioni indipendenti dall'hardware di esecuzione. Questo infatti è virtualizzato dalla piattaforma Java e ciò permette di distribuire le app in piattaforme differenti. Java può essere utilizzato nei sistemi embedded, nei dispositivi mobili, nei server enterprise e anche nei supercomputer.

Il linguaggio principale usato per generare codice che viene distribuito come byte code in una Java Virtual Machine (JVM) è Java. Nonostante ciò, altri compilatori permettono la generazione di byte code a partire da diversi linguaggi di programmazione, come JavaScript, Python o Ruby. Ci sono inoltre linguaggi che possono essere eseguiti in modo nativo nella JVM, come Scala o Apache Groovy. Per quanto riguarda la sintassi, Java ha preso molto in prestito dal C e dal C++, mentre per le caratteristiche orientate agli oggetti si è ispirata a Smalltalk e all'Objective-C. Uno dei punti fondamentali di Java è che non prevede puntatori e altri costrutti di così basso livello. Ogni oggetto creato in Java viene allocato sullo heap e tutte le variabili di tipi oggetti sono dei riferimenti. La gestione della memoria è effettuata con un meccanismo di garbage collection, così come visto in .NET.

Nel novembre del 2006, Sun Microsystem ha deciso di rendere disponibile con licensa GNU (General Public License) la sua implementazione di Java.

2.4.2.1 Principio di funzionamento

Le componenti essenziali della piattaforma Java sono la Java Virtual Machine e le API Java, cioè un insieme di librerie che permettono di svolgere qualsiasi tipo di compito. Le applicazioni software per poter essere compatibili con una piattaforma Java devono essere scritte in uno dei linguaggi supportati dalla macchina virtuale (come per esempio Java, Scala o altri) e quindi compilate in modo da ottenere il byte code che può essere interpretato ed eseguito dalla macchina virtuale. Il byte code non dipende da alcuna macchina o sistema operativo in particolare, ma è il più possibile astratto dal sistema in cui verrà poi eseguita l'applicazione. L'unica cosa che cambia in base alla tipologia della macchina è l'interprete che riceve in ingresso il byte code dell'applicazione.

Il compilatore Java ha il compito di convertire il codice sorgente Java in byte code ed è fornito dal Java Development Kit (JDK). Il Java Runtime Environment (JRE) integra la Java Virtual Machine con un compilatore just-in-time, che converte al volo il byte code in modo da ottenere codice macchina.

Dalla piattaforma Java derivano diverse altre piattaforme che si focalizzano su specifiche tipologie di dispositivi, come la Java Card (per realizzare applicazioni che eseguono su Smart card o altri dispositivi con memoria limitata), Java ME (Micro Edition, per realizzare applicazioni che eseguono su dispositivi con limitate capacità di storage, di display e di potenza), Java SE (Standard Edition, per applicazioni general-purpose sui PC o server) e Java EE (Enterprise Edition), che arricchisce Java SE con altre API per lo sviluppo di applicazioni client-server multi-tier.

2.4.2.2 Java Virtual Machine

La Java Virtual Machine si occupa di eseguire i byte code virtualizzando l'hardware al di sopra del quale viene eseguita l'applicazione java. Le specifiche che definiscono la JVM non contengono dettagli implementativi che potrebbero rappresentare un problema per garantire l'interoperabilità, come per esempio l'algoritmo di garbage collection utilizzato o eventuali ottimizzazioni interne delle istruzioni. Le applicazioni java possono essere eseguite solo utilizzando un'implementazione concreta delle specifiche astratte della Java Virtual Machine.

Per quanto riguarda l'architettura, la JVM è costituita da un heap garbage-collected in cui vengono salvati oggetti e array. Il codice, le costanti e altre classi dati sono salvati in una parte di heap che prende il nome di area dei metodi. Un'implementazione della Java Virtual Machine potrebbe trattare in modo separato l'heap e l'area dei metodi, decidendo per esempio di non effettuare garbage collection su di essa. Ogni thread della JVM ha il suo stack di chiamate, all'interno del quale vengono salvati i frame. Un frame viene creato quando viene invocato un metodo e non appena il metodo termina il frame viene eliminato. Ad ogni frame è associato uno stack degli operandi ed un array di variabili locali. Il primo è utilizzato per salvare gli operandi dei calcoli e i valori di ritorno di un metodo chiamato, mentre le variabili locali hanno lo stesso scopo dei registri.

La JVM è costituita da tre componenti fondamentali:

- Class loader:
  Serve a caricare le classi dell'applicazione Java e delle API che costituiscono il byte code. In particolare, il class loader effettua tre azioni. La prima prende il nome di

loading e consiste nel cercare e importare i dati binari per ogni tipo. Successivamente l'azione di linking verifica la correttezza del tipo importato, quindi alloca la memoria per le variabili di classe inizializzandola con dei valori di default e infine converte i riferimenti simbolici in riferimenti diretti. L'ultima fase prende il nome di inizializzazione e consiste nell'invocare quel codice java responsabile di assegnare alle variabili di classe i loro valori iniziali. È possibile distinguere tra due tipi di class loader: il bootstrap class loader e l'user defined class loader. Il primo, che si occupa di caricare le classi trusted, deve essere presente obbligatoriamente nell'implementazione di ogni JVM.

- Class verifier:
  La strategia di java è evitare che un'applicazione possa provocare il crash della macchina host o interagire in modo inappropriato con altre operazioni eseguite nella macchina host. Inoltre, java protegge metodi e strutture dati relative al codice fidato dall'accesso di codice non fidato eseguito nella JVM. In ultimo luogo java non permette che avvengano quei tipici errori dei programmatori che causano la corruzione dei dati o altri comportamenti inaspettati, come l'utilizzo di un puntatore non inizializzato o l'accesso oltre la fine di un array.
  Il class verifier è uno degli attori principali nella strategia appena mostrata e si occupa di verificare la validità del byte code. In particolare, verifica che i branch puntino sempre a degli indirizzi di memoria validi, che i dati siano sempre inizializzati assieme al fatto che i riferimenti siano type-safe e, in ultimo luogo, che l'accesso ai campi private o package all'interno delle classi sia corretto. Le prime due operazioni vengono eseguite durante il caricamento di una classe mentre la terza è eseguita in modo dinamico, cioè quando i campi di una classe sono acceduti da altre classi.
  Il verifier assicura anche che le istruzioni di branch possano puntare solo ad un'altra istruzione all'interno dello stesso metodo. Inoltre, permette che una data istruzione faccia riferimento solo ad un indirizzo di memoria fisso nello stack. Questo permette al compilatore just-in-time di convertire gli accessi allo stack in accessi ai registri senza causare una penalità in termini di velocità durante l'emulazione su un architettura basata su registri.
- Interprete java:
  L'interprete permette di eseguire ogni byte code java. Interpreti diversi sono necessari per ogni architettura hardware differente. Il problema è che quando il byte code è eseguito dall'interprete le prestazioni sono peggiori rispetto all'esecuzione del codice macchina ottenuto compilando lo stesso byte code. I compilatori just-in-time servono per superare in parte tale svantaggio. Infatti, durante l'esecuzione di un programma un compilatore del genere traduce il byte code in linguaggio macchina. Le parti di programma che vengono tradotte saranno eseguite più velocemente di quelle che invece vengono interpretate. Naturale conseguenza di quanto detto è che conviene utilizzare questa tecnica per quelle parti di programma che vengono eseguite più frequentemente.
  È possibile anche implementare l'interprete direttamente a livello hardware: in questo modo il byte code è eseguito come codice nativo. Ciò implica lo sviluppo di un processore con un set di istruzioni equivalente al repertorio di istruzioni definite dalle specifiche Sun Microsystem. Il vantaggio di un approccio del genere è che si ottengono le prestazioni migliori. D'altro lato, lo svantaggio è che il byte code può essere eseguito solo in quelle macchine che implementano java a livello hardware.

2.4.2.3 API Java

Nei sistemi operativi moderni, gli sviluppatori possono sfruttare la presenza di una grossa quantità di codice riusabile per semplificare il proprio lavoro. Questo codice è rappresentato da un insieme di dynamically linked library (dll) che possono essere caricate dalle applicazione a runtime. Il problema nel caso della piattaforma java è che essa non è associata ad uno specifico sistema operativo e quindi le applicazioni non possono utilizzare eventuali librerie fornite dall'OS. Per risolvere questo problema, la piattaforma java stessa porta con sé un insieme di librerie di classi che offrono funzionalità simili alle librerie messe a disposizione da un sistema operativo. Un esempio è dato dalla libreria Swing, responsabile di disegnare le interfacce utente e gestire gli eventi ad esse associati, nascondendo al programmatore le differenze su come diversi sistemi operativi gestiscono i propri componenti.

Le librerie di classi java hanno tre scopi principali. Il primo è lo stesso di tutte le altre librerie standard, ovvero fornire al programmatore un insieme di funzioni per eseguire le azioni più comuni, come mantenere una lista di oggetti o effettuare complesse operazioni sulle stringhe. In secondo luogo, forniscono un'interfaccia astratta per quelle operazioni che altrimenti dipenderebbero troppo dall'hardware sottostante e da un sistema operativo specifico. Compiti come l'accesso alla rete o ai file sono spesso troppo correlati alle differenti implementazioni di ogni piattaforma. Le librerie java.net e java.io implementano un livello di astrazione nel codice nativo del sistema operativo e forniscono un'interfaccia standard sulla quale le applicazioni java possono fare affidamento per compiere task che riguardano l'uso della rete e dei file. Il terzo e ultimo scopo riguarda il modo in cui viene gestita l'eventuale assenza di una o più caratteristiche che un'applicazione java si aspetta di trovare, ma che non è offerta dalla piattaforma sottostante. In questo caso le librerie di classi possono emulare un componente che fornisca la specifica non presente nell'OS oppure possono fornire un modo consistente per verificare la presenza o meno della caratteristica richiesta.

Le librerie di classi offerte sono così ampie che sono disponibili tre configurazioni diverse della piattaforma java in base all'uso che se ne vuole fare:

- Standard edition:
  Questa configurazione fornisce le API per le esigenze più comuni, permettendo di sviluppare applicazioni stand-alone, applicazioni client-server, applicazioni per accesso a database o altri tipi di applicazioni.
- Enterprise edition:
  Questa configurazione permette di scrivere applicazioni distribuite.
- Micro edition:
  Questa configurazione permette di sviluppare applicazioni per i device con limitate risorse computazionali.

2.4.2.4 Java EE

La piattaforma Java EE è una piattaforma derivata da Java ed è molto utilizzata nella programmazione web. L'insieme di specifiche fornite dalla piattaforma Java EE può essere implementato totalmente (in questo caso si usa il termine Full Platform) oppure può essere limitato al Web Profile. In quest'ultimo caso viene implementato solo quel sottoinsieme di specifiche che serve a realizzare applicazioni web.

Originariamente i software che implementavano questo insieme di specifiche venivano chiamati application server, invece oggi sono chiamati Referencing Runtime Java EE. Questo cambiamento è causato dalla nascita dell'architettura a microservizi, in cui viene eseguita un'applicazione java autonoma, senza la necessità di avere un application server. Quindi, mentre in un primo momento la specifica incitava gli sviluppatori a creare applicazioni basate su architettura multi-tier, oggi è possibile sviluppare anche microservizi.

L'insieme di specifiche che costituisce la Java EE estende le funzionalità di base della piattaforma java offrendo una varietà di tecnologie. L'utilizzo di un referencing runtime, comunque, è richiesto solo nelle componenti enterprise (che necessita di transazioni distribuite e di code distribuite). Le altre componenti, come quelle per il web o per le basi dati, possono essere utilizzate anche all'interno di un normale web server. Le componenti principali sono raggruppate nelle seguenti specifiche:

- Specifiche per i web service
  Java EE permette lo sviluppo di web service di tipo REST o di tipo SOAP. Le specifiche sono:
  - JSON Processing: serve ad elaborare le informazioni in formato JSON.
  - JSON Binding: serve per trasformare gli oggetti java in stringhe JSON e viceversa.
  - RESTful Web Services: serve per implementare web service utilizzando l'architettura REST.
  - JAX-WS: serve per realizzare web service di tipo SOAP
  - Java Architecture for XML Binding: serve per trasformare oggetti java in stringhe XML e viceversa.
- Specifiche web
  Con specifiche web si intendono quelle componenti che servono a visualizzare le pagine web. Le componenti principali sono:
  - Servlet: questa tecnologia serve a trasferire le pagine HTML al client e a esporre i web service.
  - Java Server Faces: racchiude un insieme di componenti tramite i quali si possono sviluppare siti con funzionalità AJAX
- Specifiche enterprise
  Includono quelle componenti che sono caratteristiche proprie di Java EE.
  - Contexts and Dependency Injection: permettono di usare l'inversione di controllo.
  - Enterprise JavaBeans: offrono un sistema con le specifiche richieste dalle applicazioni enterprise, quali scalabilità, sicurezza, persistenza dei dati e altro. Queste caratteristiche sono oggi disponibili anche con Contexts and Dependency Injection.
  - Java Message Service: offre un sistema per inviare e gestire messaggi.
  - Java EE Security API: serve per gestire gli utenti, le password, i gruppi e le autenticazioni.
- Specifiche per l'interazione con la base dati
  Le componenti che ricadono in questa categoria servono a gestire la persistenza dei dati all'interno dei database relazionali.
  - Java Transaction API: serve a gestire le transazioni distribuite.

- Java DataBase Connectivity: offre un'interfaccia per accedere a qualsiasi tipo di database.
- Java Persistence API: contiene le classi che vengono utilizzate per gestire la persistenza dei dati.

2.4.3 .NET vs Java

Dopo aver introdotto esaustivamente le due piattaforme principali, quella .NET e quella Java, è arrivato il momento di confrontarle per capire quali sono le similitudini e le differenze tra le due.

2.4.3.1 Similitudini tra .NET e Java

Dal punto di vista architetturale, il Common Language Runtime, il Common Intermediate Language e il linguaggio C# sono simili rispettivamente alla Java Virtual Machine, al byte code e al linguaggio Java.

Naturalmente sia .NET che Java offrono un insieme di componenti e servizi standard che semplificano lo sviluppo delle applicazioni. Questo permette agli sviluppatori di concentrarsi sulla business logic anziché perdere tempo a implementare servizi già pronti. Infatti, sia Java che .NET forniscono dei modi standard per eseguire operazioni quali l'accesso al database, la connessione a risorse remote e lo scripting delle pagine web.

Entrambe le piattaforme permettono agli sviluppatori di realizzare applicazioni con architettura multi-tier. Inoltre, entrambe sono orientate agli oggetti, offrono un sistema di tipi sicuro e includono funzionalità di garbage collection automatiche.

2.4.3.2 Differenze tra .NET e Java

Mentre .NET ha come target principale i sistemi operativi Windows (ad eccezione di Mono), Java si basa sull'idea che lo stesso software possa essere eseguito su tipi di computer differenti, senza dover essere riscritto. Questo perché le applicazioni java sono eseguite nel contesto delle Java Virtual Machine, che sono implementate per la maggior parte delle piattaforme.

Un'altra differenza è che nonostante il framework .NET sia un prodotto libero, esso è stato sviluppato da Microsoft ed integrato nei sistemi operativi Windows. Quindi non fa affidamento su alcun fornitore di terze parti nell'implementazione della piattaforma. Tuttavia, questo framework è altamente estensibile e nel corso degli anni sono state sviluppate molte estensioni ad opera di terze parti. D'altro lato, la piattaforma Java è open source, anche se ci sono diversi venditori che ne forniscono una propria implementazione. La competizione che si instaura tra i fornitori di Java è una delle forze principali di Java stessa, perché promuove l'innovazione, mentre gli standard di compatibilità garantiscono la qualità dei prodotti.

Per la piattaforma .NET Microsoft ha sviluppato un IDE standard, Microsoft Visual Studio. Esso offre tutto ciò che serve ad uno sviluppatore per quanto riguarda lo sviluppo, il debug e le operazioni di build e deploy. Dall'altro lato, non esiste un IDE standard per la piattaforma

Java e gli sviluppatori sono liberi di scegliere quello che più soddisfa i propri bisogni. Gli IDE principali in ambito Java sono Eclipse, IntelliJ Idea, Oracle NetBeans e Oracle JDeveloper.

La “Tabella 1” riassume le caratteristiche e i servizi offerti da entrambe le piattaforme.

***Tabella 1**: Comparazione tra Java e .NET*

CARATTERISTICA	MICROSOFT .NET	JAVA
Linguaggi supportati	C#, VB.NET, C++, PHP, Ruby, Python	Java, Groovy, Scala, PHP, Ruby, Python, JavaScript
Sistema operativo richiesto	Windows	Qualsiasi
Runtime	CLR	JVM
Componenti server	NET, COM	EJBs
Componenti GUI	.NET class	JavaBeans
Supporto ai web service	Built-in	Add-on
Unit testing	Microsoft Unit Testing Framework	JUnit
Framwork web app	Asp.NET MVC, Spring .NET	Spring
Scripting web server	Asp.NET	JSF
Accesso ai dati	Ado.NET	JDBC
Motore HTTP	IIS	Application Server di molteplici fornitori
Persistenza oggetti	Entity Framework, NHibernate	Hibernate

2.4.3.3 Come scegliere tra .NET e Java

Senza dubbio entrambe le piattaforme permettono di sviluppare applicazioni enterprise di elevata qualità. Tuttavia, la scelta di una tecnologia non ricade solo su quanto viene offerto in termini di performance.

La piattaforma .NET sembra essere avvantaggiata nello sviluppo di applicazioni basate su interfacce utente molto ricche. D’altro lato, Java rappresenta una soluzione migliore per le applicazioni che devono supportare un elevato numero di utenti. In secondo luogo, se

l’applicazione sviluppata deve essere portabile su sistemi diversi conviene puntare su Java, che è multipiattaforma per natura.

Nel caso in cui la scelta sia a livello aziendale, bisogna considerare qual è la tecnologia che è già predominante all’interno dell’azienda, poiché rimpiazzare le infrastrutture potrebbe essere costoso. Allo stesso modo bisogna tenere in conto le skill degli sviluppatori a disposizione, perché si potrebbero affrontare costi elevati per la formazione dei developer nel caso in cui si voglia cambiare tecnologia.

3. Requisiti dell'applicazione

L'obbiettivo di questo elaborato di tesi è la progettazione, l'analisi e la realizzazione di un'applicazione web per la gestione di progetti software con front-end responsive in Angular e Bootstrap.

L'applicazione ha lo scopo di aiutare ad organizzare il lavoro dei team di sviluppo e di monitorare lo stato di avanzamento dei progetti chiavi in mano. Tramite questa applicazione si possono gestire le anagrafiche delle risorse, delle commesse, dei progetti, dei task dei progetti e dei profili di utenza e si possono anche assegnare task a specifiche risorse. Inoltre, permette anche la consuntivazione giornaliera delle risorse con l'aggiornamento dello stato di avanzamento dei vari task. Infine, offre anche la possibilità di generare dei report sullo stato di avanzamento dei progetti, sia in giorni rispetto a quelli previsti sia sotto il profilo economico.

3.1 Funzionalità previste

Le funzionalità fornite dall'applicazione sono elencate nella "Tabella 2".

***Tabella 2**: Elenco funzionalità applicazione*

Funzionalità	Descrizione
Anagrafica commessa	Censimento di tutte le commesse utili
Anagrafica progetto	Censimento di tutti i progetti, con legame con l'offerta e con la commessa
Anagrafica risorse	Censimento di tutte le risorse interessate, anche di quelle che non avranno accesso all'applicazione
Macro-attività	Gestione della tabella delle macro-attività
Profilo	Gestione della tabella di dominio dei profili
Tipi risorsa	Gestione della tabella di dominio dei tipi risorsa
Stato consuntivo	Gestione della tabella di dominio degli stati consuntivi
Associazione progetto\risorse	Permette di associare un progetto ad una o più risorse

Associazione progetto\task	Permette di associare i task ad un progetto e assegnarli alle risorse del progetto
Consuntivazione	Permette di gestire la consuntivazione giornaliera delle attività svolte da ciascuna risorsa
Dashboard	Permette di conoscere l'andamento delle attività progettuali (BDG, ACT, ETC…) a livello di singolo task, progetto o commessa

3.2 Definizione interfaccia utente

Nella fase di definizione dell'interfaccia utente è utile fare uso di una mappa di navigazione che descrive la struttura dell'applicazione ed in particolare mostra come sono disposte le varie web page. La "Figura 4" mostra la mappa di navigazione dell'applicazione sviluppata.

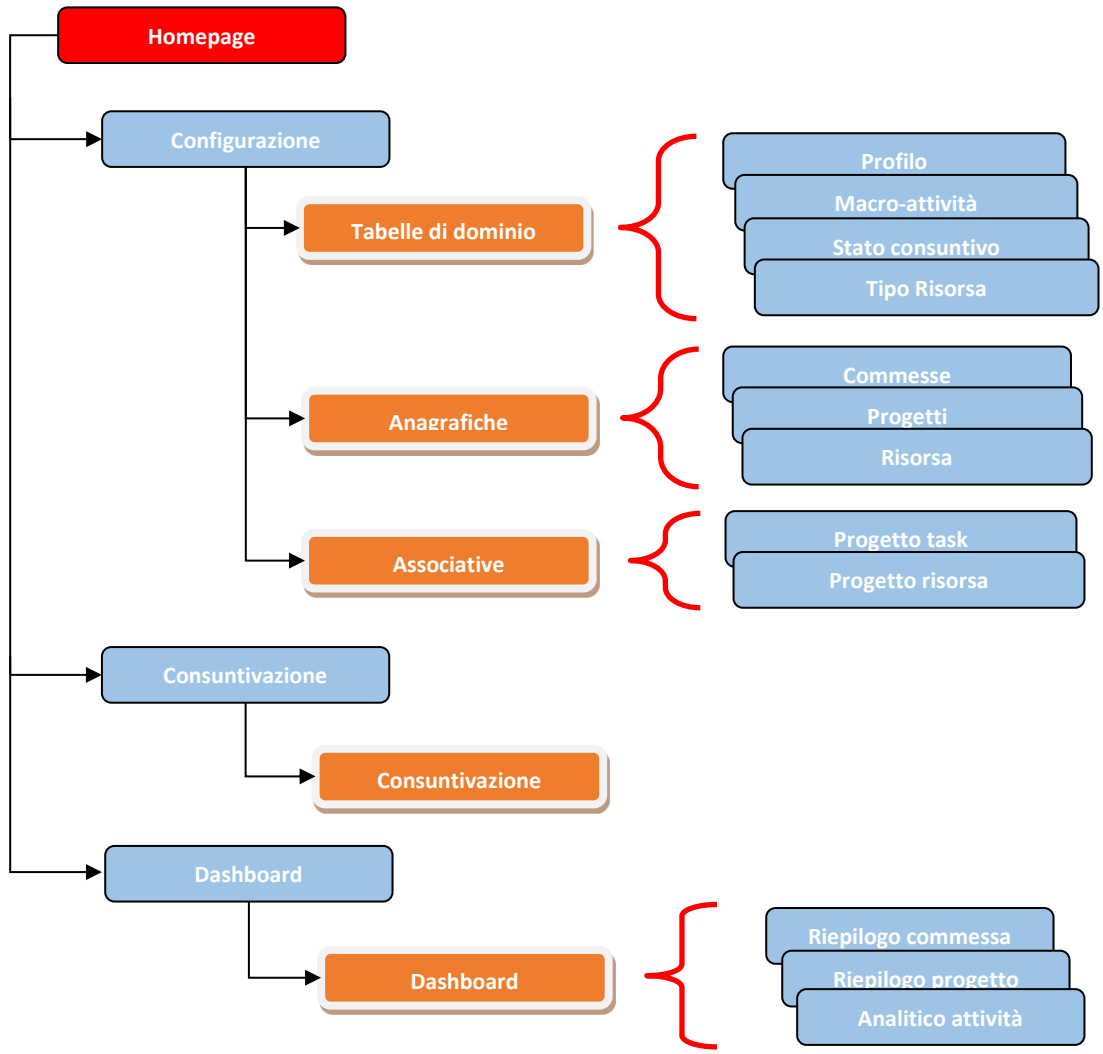
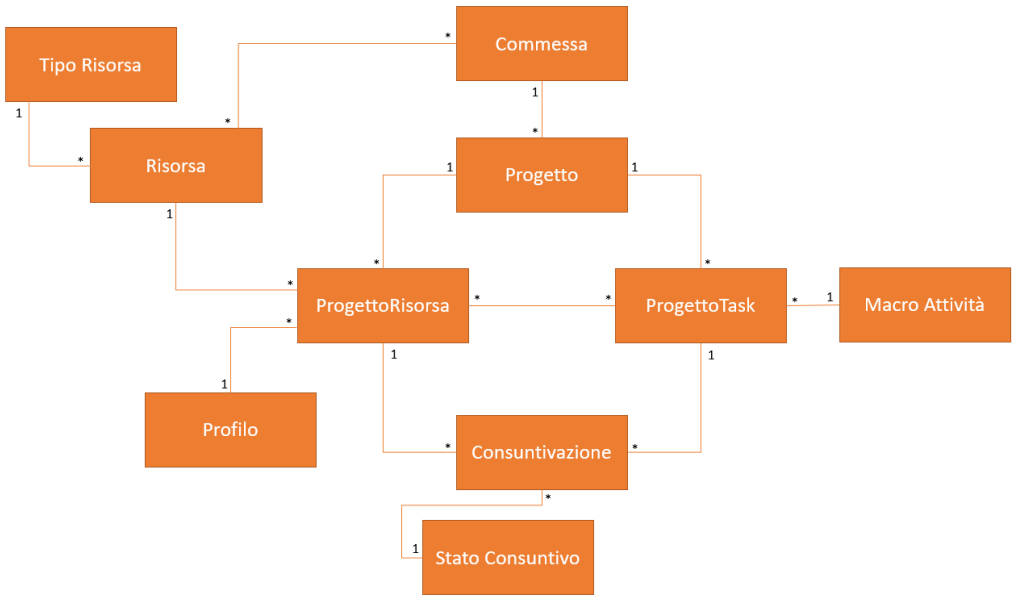


***Figura 4**: Mappa di navigazione dell'applicazione*

Al caricamento dell'applicazione l'utente si troverà di fronte alla homepage. Il menù è costituito da tre voci principali: configurazione, consuntivazione e dashboard. A sua volta configurazione contiene un sottomenù con altre tre voci che permettono all'utente di muoversi tra le tabelle di dominio, le anagrafiche e le associative. Nella pagina delle tabelle di dominio è possibile analizzare i profili, gli stati consuntivi, le macro-attività e i tipi risorsa. Nella pagina delle anagrafiche è possibile consultare le commesse, i progetti e le risorse. Nella pagina delle associative si possono analizzare le relazioni tra un progetto e le risorse ad esso assegnate e tra un progetto e i suoi task, con le risorse impegnate in ciascuno di essi. All'interno della sezione dashboard sarà possibile consultare dei report sull'andamento della commessa, dei progetti o dei task associati ai vari progetti. La pagina relativa alla consuntivazione permette alle risorse di consuntivare le attività che vengono svolte giornalmente.

3.3 Modello del dominio

I domain model servono a mostrare le classi del dominio, gli attributi (cioè le proprietà delle classi) e le relazioni (cioè i collegamenti tra le varie classi). Il modello del dominio dell'applicazione sviluppata è mostrato nella "Figura 5".



Attributi:

Commessa: short IDCommessa, string CodCommessa, string DescCommessa, bool FlgCestino, List ListaProgetti, List ResponsabiliCommessa

Progetto: short IDProgetto, string CodProgetto, string DescProgetto, short IDCommessa, string CodOfferta, DateTime Inizio, DateTime Fine, decimal budget, bool FlgCestino, List ListaProgettoTask, List ListaProgettoRisorse

MacroAttivita: short IDMacroAttivita, string DescMacroAttivita, bool FlgCestino

TipoRisorsa: short IDTipoRisorsa, string TipologiaRisorsa, bool FlgCestino

Risorsa: short IDRisorsa, string Matricola, string Nome, string Cognome, short IDTipoRisorsa, string CodNavision, decimal OreSettimanali, DateTime DataInizio, DateTime DataFIne, bool FlgCestino

Profilo: short IDProfilo, string DescProfilo, bool FlgCestino

ProgettoTask: short IDProgettoTask, short IDProgetto, short IDMacroAttivita, string DescTask, decimal Budget, short Ordine, string DescEstesa, bool FlgCestino, List ListaProgettoRisorsa, List ListaConsuntivazioni

ProgettoRisorsa: short IDProgettoRisorsa, short IDProgetto, short IDRisorsa, short IDProfilo, decimal ActCost, decimal ActDay, bool FlgCestino, List ListaProgettoTask

StatoConsuntivo: short IDStatoConsuntivo, string DescStatoConsuntivo, bool FlgCestino

Consuntivazione: short IDConsuntivazione, DateTime IDGiorno, short IDProgettoTask, short IDProgettoRisorsa, short IDStatoConsuntivo, decimal Actual, decimal Etc, string Note, bool FlgCestino

***Figura 5**: domain model dell'applicazione sviluppata*

3.3.1 Tabelle di dominio

Come mostrato precedentemente nella "Figura 4" tra le tabelle di dominio rientrano il Profilo, lo Stato Consuntivo, la Macro-attività e il Tipo Risorsa.

3.3.1.1 Profilo

La funzionalità permette di gestire i vari profili. I profili censiti sono:

- Amministratore
- Responsabile
- Developer

Tramite questa funzionalità è possibile inserire un nuovo profilo e modificare o eliminare un profilo già esistente. La cancellazione non è mai fisica ma esclusivamente logica, impostando l'attributo FlgCestino dell'oggetto Profilo in esame a 1.

La pagina associata a questa funzionalità contiene una griglia con i profili già censiti, comprensiva di pulsanti per inserire un nuovo profilo e modificare o cancellarne uno esistente. La griglia inoltre offre le funzionalità di filtro dei dati e export in Excel. Nel caso di inserimento o modifica si apre un pop-up contenente il campo "Descrizione profilo" e dei pulsanti per chiudere o salvare quanto inserito dall'utente.

Il profilo di responsabile può essere inteso da due punti di vista differenti: responsabile di commessa e responsabile di progetto.

I livelli autorizzativi per il profilo sono elencati nella "Tabella 3".

***Tabella 3**: Livelli autorizzativi della tabella di dominio Profilo*

Profilo	Inserisce profilo	Modifica profilo	Elimina profilo	Consulta profili
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	NO	NO	NO	NO
Responsabile (progetto)	NO	NO	NO	NO
Developer	NO	NO	NO	NO

3.3.1.2 Macro-attività

La funzionalità permette di gestire le varie macro-attività. Le macro-attività censite sono:

- Analisi
- Sviluppo

- Test
- Deploy

Tramite questa funzionalità è possibile inserire una nuova macro-attività e modificare o eliminare una macro-attività già esistente. La cancellazione non è mai fisica ma esclusivamente logica, impostando l'attributo FlgCestino dell'oggetto MacroAttivita in esame a 1.

La pagina associata a questa funzionalità contiene una griglia con le macro-attività già censite, comprensiva di pulsanti per inserirne una nuova e modificare o cancellarne una esistente. La griglia inoltre offre le funzionalità di filtro dei dati e export in Excel. Nel caso di inserimento o modifica si apre un pop-up contente il campo "Descrizione macro-attività" e dei pulsanti per chiudere o salvare quanto inserito dall'utente.

I livelli autorizzativi per la macro-attività sono elencati nella "Tabella 4".

***Tabella 4**: Livelli autorizzativi della tabella di dominio Macro-attività*

Profilo	Inserisce Macro-attività	Modifica Macro-attività	Elimina Macro-attività	Consulta Macro-attività
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	NO	NO	NO	NO
Responsabile (progetto)	NO	NO	NO	NO
Developer	NO	NO	NO	NO

3.3.1.3 Stato consuntivo

La funzionalità permette di gestire i vari stato consuntivo. Gli stati censiti sono:

- Aperto
- Chiuso
- Validato

Tramite questa funzionalità è possibile inserire un nuovo stato consuntivo e modificare o eliminare uno stato già esistente. La cancellazione anche in questo caso non è mai fisica ma esclusivamente logica, impostando l'attributo FlgCestino dell'oggetto StatoConsuntivo in esame a 1.

La pagina associata a questa funzionalità contiene una griglia con gli stati consuntivi già censiti, comprensiva di pulsanti per inserirne uno nuovo e modificare o cancellarne uno esistente. La griglia inoltre offre le funzionalità di filtro dei dati e export in Excel. Nel caso di inserimento o modifica si apre un pop-up contente il campo "Descrizione stato consuntivo" e dei pulsanti per chiudere o salvare quanto inserito dall'utente.

I livelli autorizzativi per lo stato consuntivo sono elencati nella “Tabella 5”.

***Tabella 5**: Livelli autorizzativi della tabella di dominio Stato consuntivo*

Profilo	Inserisce stato consuntivo	Modifica stato consuntivo	Elimina stato consuntivo	Consulta stato consuntivo
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	NO	NO	NO	NO
Responsabile (progetto)	NO	NO	NO	NO
Developer	NO	NO	NO	NO

3.3.1.4 Tipo risorsa

La funzionalità permette di gestire i vari tipi risorsa. I tipi risorsa censiti sono:

- Risorsa interna
- Fornitore esterno
- Stagista

Questa funzionalità permette di inserire un nuovo tipo risorsa e di modificare o eliminare un tipo risorsa già esistente. La cancellazione è esclusivamente logica ed è ottenuta impostando a 1 l’attributo FlgCestino del tipo risorsa in esame.

La pagina associata a questa funzionalità mostra una griglia con i tipi risorsa già censiti e include dei bottoni per inserirne uno nuovo e modificare o eliminarne uno esistente. Anche in questo caso la griglia offre le funzionalità di filtro dei dati ed export in Excel. Nel caso di inserimento o modifica viene mostrato un popup contenente il campo “Tipologia risorsa” e dei pulsanti per annullare l’operazione o salvare quanto inserito dall’utente.

I livelli autorizzativi per questa funzionalità sono mostrati nella “Tabella 6”.

***Tabella 6**: Livelli autorizzativi della tabella di dominio Tipo risorsa*

Profilo	Inserisce tipo risorsa	Modifica tipo risorsa	Elimina tipo risorsa	Consulta tipo risorsa
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	NO	NO	NO	NO

Responsabile (progetto)	NO	NO	NO	NO
Developer	NO	NO	NO	NO

3.3.2 Anagrafiche

All'interno della sezione dedicata alle anagrafiche si trovano le anagrafiche delle commesse, quelle dei progetti e quelle delle risorse.

3.3.2.1 Commessa

Questa funzionalità permette di gestire le anagrafiche commesse. In particolare, permette di inserire una nuova commessa e modificare o eliminare una commessa già esistente. La cancellazione non è mai fisica ma esclusivamente logica e avviene impostando ad 1 la proprietà FlgCestino dell'oggetto commessa da rimuovere. Quando si decide di eliminare una commessa, devono essere cancellati anche tutti i progetti associati ad essa.

La pagina che implementa questa funzionalità contiene una griglia con l'elenco di tutte le commesse che sono già state censite, con dei pulsanti per inserirne una nuova e modificare o eliminarne una esistente. La griglia inoltre offre la funzionalità di filtro dei dati ed export degli stessi in Excel. Nel caso di inserimento o modifica si apre un pop-up con i campi "Codice commessa", "Descrizione commessa", "Responsabili" e dei pulsanti annullare o salvare quanto inserito dall'utente. Il campo responsabile permette una selezione multipla di risorse che vengono elette a responsabili della commessa.

I livelli autorizzativi per le anagrafiche commesse sono elencati nella "Tabella 7".

***Tabella** 7: Livelli autorizzativi delle anagrafiche Commesse*

Profilo	Inserisce commessa	Modifica commessa	Elimina commessa	Consulta commessa
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	NO	SI (solo le commesse per le quali è responsabile)	NO	SI (solo le commesse per le quali è responsabile)
Responsabile (progetto)	NO	NO	NO	SI (solo le commesse per le quali è responsabile di almeno un progetto)
Developer	NO	NO	NO	NO

3.3.2.2 Progetto

Questa funzionalità permette di gestire le anagrafiche progetti. In particolare, permette di censire un nuovo progetto e modificare o eliminare un progetto già esistente. La cancellazione non è mai fisica ma esclusivamente logica e si ottiene impostando ad 1 la proprietà FlgCestino dell'oggetto progetto che deve essere eliminato. Questa operazione deve essere consentita solo al profilo di Amministratore e a quello di Responsabile.

La pagina associata a questa funzionalità contiene una griglia con i progetti che sono già stati censiti, comprensiva di pulsanti per l'inserimento di un nuovo progetto e la modifica o cancellazione di un progetto già esistente. Inoltre, la griglia permette anche di filtrare i dati e di effettuare un export in Excel. Nel caso di inserimento o modifica viene mostrato un pop-up contenente i campi "Codice progetto", "Descrizione progetto", "Codice commessa", "Codice offerta", "Budget (espresso in giorni)", "Data di inizio", "Data di fine" e dei pulsanti per la chiusura o il salvataggio.

I livelli autorizzativi per le anagrafiche progetti sono mostrati nella "Tabella 8".

***Tabella 8**: Livelli autorizzativi delle anagrafiche Progetti*

Profilo	Inserisce progetto	Modifica progetto	Elimina progetto	Consulta progetto
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	SI (solo se è responsabile della commessa scelta)	SI (solo i progetti che appartengono alle commesse di cui è responsabile)	SI (solo i progetti che appartengono alle commesse di cui è responsabile)	SI (solo i progetti che appartengono alle commesse di cui è responsabile)
Responsabile (progetto)	NO	SI (solo i progetti per i quali è responsabile)	SI (solo i progetti per i quali è responsabile)	SI (solo i progetti per i quali è responsabile)
Developer	NO	NO	NO	NO

3.3.2.3 Risorsa

La funzionalità permette di gestire le anagrafiche risorse. In particolare, permette di inserire una nuova risorsa e modificare o eliminare una risorsa già esistente. Anche in questo caso la cancellazione non è mai fisica, ma esclusivamente logica e si ottiene impostando ad 1 la proprietà FlgCestino della risorsa che deve essere cancellata.

La pagina che implementa questa funzionalità contiene una griglia con l'elenco delle risorse già censite con dei pulsanti che permettono di inserire una nuova risorsa e modificare o cancellarne una già esistente. Anche in questo caso è possibile filtrare i dati ed effettuare un export in Excel. Nel caso di inserimento o modifica si apre un pop-up contenente i campi

"Matricola", "Cognome", "Nome", "Profilo", "Tipo risorsa", "CodNavision", "Ore settimanali", "Data inizio", "Data fine", più dei pulsanti "Chiudi" e "Salva".

I livelli autorizzativi associati a questa funzionalità sono elencati nella "Tabella 9", mentre la "Figura 6" mostra le stesse informazioni nella forma dei casi d'uso UML.

***Tabella 9**: Livelli autorizzativi delle anagrafiche Risorse*

Profilo	Inserisce risorsa	Modifica risorsa	Elimina risorsa	Consulta risorsa
Amministratore	SI	SI	SI	SI
Responsabile (commessa)	SI	SI	NO	SI
Responsabile (progetto)	SI	SI	NO	SI
Developer	NO	NO	NO	NO

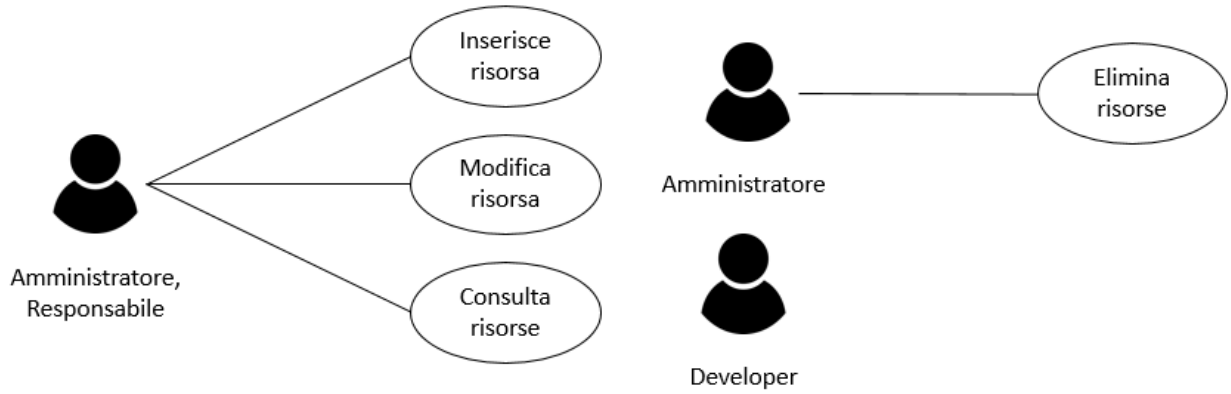


***Figura 6**: Use case anagrafica Risorse*

3.3.3 Associative

Le associative permettono di associare entità diverse del dominio tra di loro. Tra queste ricadono l'associazione progetto-task e l'associazione progetto-risorsa.

3.3.3.1 Progetto-Task

Questa funzionalità permette di aggiungere un task di lavoro ad un progetto e modificare o annullare un'associazione del genere. La cancellazione non è mai fisica, ma è esclusivamente logica ed è ottenuta impostando ad 1 il valore della proprietà FlgCestino. Questa operazione può essere effettuata soltanto dai profili Amministratore e Responsabile: il primo può operare

su tutti i progetti mentre il secondo può agire solo sui progetti di cui è responsabile o che appartengono ad una commessa di cui è responsabile.

La pagina che offre questa funzionalità contiene una dropdown che mostra tutti i progetti attivi. Alla scelta di un progetto viene visualizzata una griglia che contiene l'elenco di tutti i task attivi ad esso associati, comprensiva di pulsanti per inserire un nuovo task e modificare o eliminare un task già esistente. Inoltre, la griglia offre anche le funzionalità di filtro dei dati e di export in Excel. Nel caso di inserimento o modifica viene mostrato un pop-up contenente i campi "Progetto" (preselezionato al progetto scelto precedentemente), "Macro-attività", "Descrizione Task" (univoca all'interno del progetto), "Budget" (in giorni), "Ordine", "Descrizione estesa", "Risorse", più dei pulsanti per annullare l'operazione chiudendo il pop-up o effettuare il salvataggio dei dati inseriti dall'utente. Il campo "Macro-attività" elenca tutte le macro-attività attive e permette la selezione puntuale. Il campo "Risorse" elenca tutte le risorse che sono state associate precedentemente al progetto e permette la selezione puntuale o multipla. Nel caso in cui l'utente prova ad inserire un task con un budget che provoca lo sforamento del budget totale associato al progetto, viene segnalato un warning e il salvataggio dei dati viene effettuato ugualmente. Il pop-up per l'inserimento/modifica di un'associazione del genere mostra anche il budget totale del progetto scelto, il budget già assegnato e il budget ancora disponibile.

I livelli autorizzativi per questa funzionalità sono mostrati nella "Tabella 10", mentre la "Figura 7" mostra le stesse informazioni sotto forma di casi d'uso UML.

***Tabella 10**: Livelli autorizzativi dell'associativa Progetto-Task*

Profilo	Associa progetto-task	Modifica progetto-task	Elimina progetto-task	Consulta progetto-task
Administrator	SI	SI	SI	SI
Responsabile (commessa)	SI (solo i progetti appartenenti a commesse di cui è responsabile)	SI (solo i progetti appartenenti a commesse di cui è responsabile)	SI (solo i progetti appartenenti a commesse di cui è responsabile)	SI (solo i progetti appartenenti a commesse di cui è responsabile)
Responsabile (progetto)	SI (solo i progetto di cui è responsabile)	SI (solo i progetto di cui è responsabile)	SI (solo i progetto di cui è responsabile)	SI (solo i progetto di cui è responsabile)
Developer	NO	NO	NO	NO

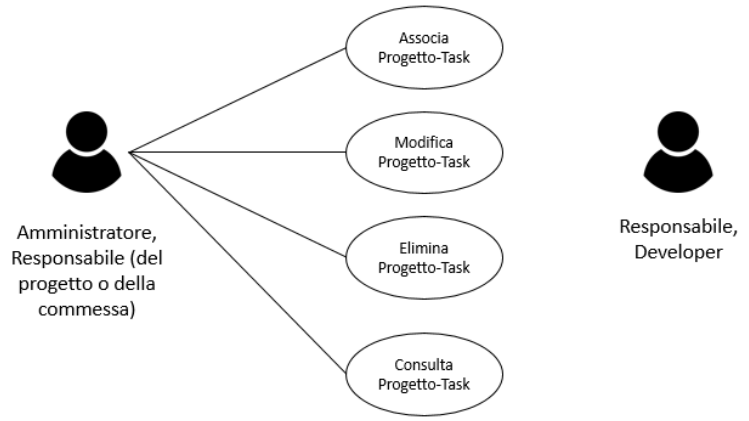


***Figura** 7: Use case Progetto-Task*

3.3.3.2 Progetto-Risorse

Questa funzionalità permette di associare un progetto ad una o più risorse e modificare o eliminare un'associazione esistente. La cancellazione è esclusivamente logica ed è ottenuta impostando ad 1 il valore della proprietà FlgCestino dell'associazione Progetto-risorsa in esame.

La pagina associata a questa funzionalità contiene una dropdown che mostra tutti i progetti attivi. Selezionato il progetto di interesse viene visualizzata una griglia con tutte le risorse attive ad esso associate, con dei pulsanti per inserire una nuova risorsa e modificare o eliminare un'associazione esistente. Inoltre, è possibile filtrare i dati o effettuare un export in Excel. Nel caso di inserimento o modifica viene visualizzato un pop-up che contiene i campi "Progetto" (preselezionato al progetto scelto precedentemente), "Risorsa", "Profilo", "ACT_cost" (costo orario della risorsa) e "ACT_day" (impegno della risorsa espresso in ore) e dei pulsanti per annullare l'operazione o salvare i dati inseriti. Il campo risorsa elenca tutte le risorse attive e permette la selezione puntuale.

La "Tabella 11" elenca i livelli autorizzativi per la funzionalità che regola le associazioni tra progetti e risorse.

***Tabella 11**: Livelli autorizzativi dell'associativa Progetto-Risorsa*

Profilo	Associa progetto-risorse	Modifica progetto-risorse	Elimina progetto-risorse	Consulta progetto-risorse
Administrator	SI	SI	SI	SI
Responsabile (commessa)	SI (solo i progetti appartenenti a commesse di cui è responsabile)	SI (solo i progetti appartenenti a commesse di cui è responsabile)	SI (solo i progetti appartenenti a commesse di cui è responsabile)	SI (solo i progetti appartenenti a commesse di cui è responsabile)

Responsabile (progetto)	SI (solo i progetto di cui è responsabile)	SI (solo i progetto di cui è responsabile)	SI (solo i progetto di cui è responsabile)	SI (solo i progetto di cui è responsabile)
Developer	NO	NO	NO	NO

3.3.4 Consuntivazione

Questa funzionalità permette a ciascuna risorsa allocata sui progetti di gestire la consuntivazione delle attività svolte.

Per procedere alla consuntivazione occorre selezionare il progetto di riferimento.

A fronte del progetto scelto viene visualizzata una griglia con il dettaglio di quanto è già stato consuntivato. I dati visualizzati sono la data della consuntivazione, il progetto, il task, la risorsa, gli actual (le ore spese sull'attività), l'etc (estimated time to complete), lo stato consuntivo e le note. La griglia offre anche la funzionalità di filtro dei dati e di export in Excel e comprende dei pulsanti per l'inserimento di una nuova consuntivazione e la modifica o cancellazione di una consuntivazione già esistente. Nel caso di inserimento o modifica viene mostrato un pop-up che contiene i campi "Giorno" (preselezionato al giorno corrente", "Progetto" (preselezionato al progetto scelto precedentemente), "Risorsa" (disabilitato nel caso in cui chi consuntiva ha il ruolo di developer all'interno del progetto scelto), "Task", "Ore consuntivate", "Etc" (espressa in giorni e ore), "Stato" (nascosto nel caso in cui chi consuntiva ha il ruolo di developer all'interno del progetto scelto), "Note" e dei pulsanti per annullare l'operazione o per effettuare il salvataggio dei dati inseriti. Nel caso in cui l'inserimento o la modifica siano effettuate da un developer, il campo task elenca solo i task ai quali la risorsa è allocata. Chi consuntiva deve avere anche la possibilità di consuntivare sui task con etc pari a 0, cioè su quei task completati. L'operazione è facilitata da uno switch che permette di mostrare nel campo task o tutti i task oppure solo quelli non completi. Il pop-up per l'inserimento/modifica di una consuntivazione mostra anche il totale delle ore che sono già state consuntivate per il task scelto e il budget associato al tast stesso.

I livelli autorizzativi per la funzionalità appena discussa sono mostrati nella "Tabella 12".

***Tabella 12**: Livelli autorizzativi per la consuntivazione*

Profilo	Inserisci cons.	Modifica cons.	Elimina cons.	Consulta cons.	Chiusura cons.	Approva cons.
Amm.	SI	SI	SI	SI	SI	SI
Resp. (commessa)	SI (solo per i progetti appartenenti a commesse di cui è responsabile)	SI (solo per i progetti appartenenti a commesse di cui è responsabile)	SI (solo per i progetti appartenenti a commesse di cui è responsabile)	SI (solo per i progetti appartenenti a commesse di cui è responsabile)	SI (solo per i progetti appartenenti a commesse di cui è responsabile)	SI (solo per i progetti appartenenti a commesse di cui è responsabile)

Resp. (progetto)	SI (solo per i progetti di cui è responsabile)	SI (solo per i progetti di cui è responsabile)	SI (solo per i progetti di cui è responsabile)	SI (solo per i progetti di cui è responsabile)	SI (solo per i progetti di cui è responsabile)	SI (solo per i progetti di cui è responsabile)
Developer	SI (solo per i progetti di cui si è developer)	SI (solo per i progetti di cui si è developer)	SI (solo per i progetti di cui si è developer)	SI (solo per i progetti di cui si è developer)	NO	NO

3.3.5 Dashboard

La funzionalità permette di conoscere in tempo reale l'andamento delle attività di progetto.

Sono previsti tre livelli di dettaglio: il primo livello è la Commessa, il secondo è il progetto e il terzo sono le attività associate ad un progetto. Gli indicatori di commessa sono ottenuti sommando i valori di tutti i progetti associati alla commessa. Gli indicatori di progetto sono ottenuti sommando i valori di tutte le attività associate al progetto. Ciò implica che la sequenza di calcolo di tutti gli indicatori avvenga nell'ordine attività, progetto, commessa.

Gli indicatori che vengono mostrati per ognuno dei tre livelli all'interno della dashboard sono il totale delle ore consuntivate, il budget a disposizione e l'etc (estimated time to complete).

È possibile filtrare i dati visualizzati nella dashboard in due modi: tramite uno switch che permette di presentare o meno i progetti finiti (cioè con etc pari a zero) oppure selezionando un range di date che permette di visualizzare solo quei progetti la cui data di inizio ricade all'interno di questo range.

I livelli autorizzativi per la funzionalità appena descritta sono mostrati nella "Tabella 13".

***Tabella 13**: Livelli autorizzativi per la dashboard*

Profilo	Visualizza dashboard
Amministratore	SI
Responsabile (commessa)	SI
Responsabile (progetto)	SI
Developer	SI

4. Strumenti utilizzati per la realizzazione dell'applicazione

Nel corso di questo capitolo verranno descritti gli strumenti che sono stati scelti per lo sviluppo dell'applicazione, assieme ai vantaggi o ai motivi che hanno portato alla loro scelta. Prima però, verrà fornita una panoramica su quelli che possono essere considerati i requisiti non funzionali per lo sviluppo di un software di qualità.

Per quanto riguarda il back-end si è deciso di puntare sulle Web Api 2 offerte dal framework Asp.NET, mentre per quanto riguarda il front-end è stata utilizzata la versione più recente del framework Angular. Inoltre, si è fatto largo utilizzo di Kendo UI for Angular, una libreria di componenti per la UI molto potenti, che fornisce funzionalità avanzate riducendo il tempo di sviluppo delle web application. Per la persistenza dei dati è stato scelto SQL Server e per effettuare il mapping object-relational è stato utilizzato l'Entity Framework con approccio Code First.

4.1 Requisiti non funzionali

Durante lo sviluppo software è utile basarsi su una serie di principi, o regole informali, che possono aiutare a migliorare la qualità del software sviluppato. Non si parla quindi di quello che fa un programma, ma di come lo fa.

Tra queste regole informali, quelle che possono essere considerate fondamentali sono la manutenibilità, l'affidabilità, l'usabilità e l'efficienza del codice sviluppato.

4.1.1 Manutenibilità e i cinque principi SOLID

Con manutenibilità si indica la facilità con cui un prodotto può essere modificato in modo da correggere i bug presenti sin dall'inizio, correggere i bug introdotti da precedenti modifiche, introdurre nuove funzionalità, migliorare la qualità del software, migliorare le funzionalità già implementate, adattarlo ai cambiamenti dell'hardware o del sistema operativo sottostante.

Il termine SOLID è un acronimo mnemonico che si riferisce a cinque principi che permettono di sviluppare software mantenibile e facilmente estendibile. Questi principi sono stati descritti da Robert C. Martin in una serie di articoli pubblicati nel corso dei primi anni 2000.

I cinque principi SOLID sono il principio di singola responsabilità, il principio aperto/chiuso, il principio di sostituzione di Liskov, il principio di segregazione delle interfacce e il principio di inversione delle dipendenze.

Il principio di singola responsabilità afferma che ogni elemento di un programma, come per esempio una classe o un modulo, dovrebbe essere responsabile di una singola funzionalità del software, e questa responsabilità dovrebbe essere incapsulata interamente all'interno dell'elemento stesso. Questo vuol dire che un elemento dovrebbe avere un solo motivo per dover essere modificato.

Il principio aperto/chiuso afferma che ogni elemento di un programma dovrebbe essere aperto alle estensioni, ma chiuso alle modifiche: in questo modo è possibile modificare il suo comportamento senza dover cambiare il suo codice sorgente. L'ereditarietà (dell'implementazione o dell'interfaccia) permette di soddisfare tale principio, prestando attenzione a non violare quello di singola responsabilità e quello di sostituzione di Liskov.

Il principio di sostituzione di Liskov afferma che si dovrebbe poter sostituire un oggetto con un suo sottotipo senza modificare la correttezza del programma che li utilizza.

Il principio di segregazione delle interfacce afferma che un client non dovrebbe dipendere da interfacce che contengono metodi che il client non usa. Conviene quindi avere più interfacce specifiche che una singola interfaccia generica, così che il client conosca solo i metodi che utilizza veramente.

Il principio dell'inversione delle dipendenze afferma che una classe dovrebbe dipendere da astrazioni e non da classi concrete. Questo principio permette di disaccoppiare i moduli software, perché evita che i moduli di alto livello dipendano dai moduli di basso livello e fa in modo che entrambi dipendano da astrazioni. Infatti, quello che accade normalmente nella programmazione è strutturare il software in une gerarchia di componenti, in cui quelli di alto livello sfruttano i componenti del livello sottostante per realizzare le proprie funzioni. Questa dipendenza si ripercuote anche nella compilazione. Invece, seguendo questo principio si fa in modo che i componenti di alto livello dipendano da astrazioni riguardo il funzionamento dei componenti di basso livello: queste astrazioni sono le stesse a cui fanno riferimento i componenti di basso livello. La differenza sta nel fatto che il componente di alto livello utilizza queste astrazioni per realizzare le proprie funzioni, mentre quello di basso livello le implementa. In questo modo, all'atto della compilazione non si genera alcuna dipendenza di compilazione tra i componenti di alto livello e quelli di basso livello. Per rispettare questo principio si utilizzano normalmente le interfacce o le classi astratte in abbinamento a tecniche quali la dependency injection.

Sviluppare codice mantenibile vuol dire anche sviluppare codice che sia facilmente comprensibile da altri sviluppatori, testabile e riusabile nello sviluppo di nuovi software.

4.1.2 Affidabilità

Per affidabilità si intende la frequenza con cui si verificano malfunzionamenti. Meno frequenti sono, più il software è affidabile. In altre parole si potrebbe dire che l'affidabilità indica quanto un utente può fidarsi del software che sta utilizzando.

Nel caso di sviluppo di applicazioni non critiche, che quindi non possono causare danni alla salute degli utenti che ne fanno uso, il loro rilascio iniziale avviene con una frequenza di malfunzionamenti abbastanza elevata. Infatti, ciò è vantaggioso dal punto di vista economico, perché eventuali errori scoperti dagli utenti e segnalati al produttore vengono corretti tramite il rilascio di patch periodiche.

Un concetto correlato a quello di affidabilità è la robustezza, che indica come il software si comporta di fronte a situazioni impreviste che non erano state considerate nelle specifiche, come per esempio l'introduzione di dati in input scorretti, o il fallimento di componenti software o hardware esterni.

4.1.3 Efficienza

Con il termine efficienza si indica la capacità di offrire certe prestazioni utilizzando memoria e CPU senza sprechi, ma in modo proporzionale ai servizi svolti.

4.1.4 Usabilità

Per usabilità si intende l'efficacia, l'efficienza e la soddisfazione con cui un utente può raggiungere un certo obiettivo all'interno di un contesto di utilizzo ben determinato. Il termine efficacia indica se l'utente ha raggiunto il suo obiettivo in modo preciso e completo. Il termine efficienza indica le risorse che sono servite per raggiungere uno specifico obiettivo con una certa precisione e completezza. Il termine soddisfazione indica la possibilità di raggiungere un certo obiettivo senza disagio e con attitudine positiva. Quindi lo scopo è quello di ridurre lo sforzo cognitivo dell'utente durante le fasi di comprensione e utilizzo del prodotto.

4.2 Front-end: Angular 5

Angular è un framework open source per lo sviluppo di applicazioni web. Il suo predecessore era AngularJS ed entrambi sono stati sviluppati prevalentemente da Google. Il primo rilascio di Angular è stato effettuato il 14 settembre 2016.

Angular ed AngularJS non sono compatibili ed il primo è stato riscritto per intero rispetto al secondo utilizzando un linguaggio di programmazione differente: quello di AngularJS è JavaScript, mentre quello di Angular è TypeScript.

La caratteristica principale delle web app realizzate in Angular è che l'applicazione viene scaricata dal server e poi viene eseguita completamente dal web browser. In questo modo quando l'utente richiede un'azione non c'è la necessità di rimandare la web page al server. Le applicazioni Angular possono girare in tutti i browser moderni quali Chrome, Firefox, Safari, Edge, Internet Explorer e Opera.

Con Angular diventa molto più semplice realizzare applicazioni destinate non solamente ad uno uso desktop, ma anche mobile. Infatti, combinando Angular con Bootstrap l'applicazione diventa responsive e quindi il layout si adatta alla dimensione del display in cui viene eseguita.

4.2.1 TypeScript

TypeScript è un linguaggio di programmazione open-source ed è stato sviluppato da Microsoft. Poiché TypeScript è un super-set di JavaScript, i programmi validi in JavaScript sono validi anche in TypeScript senza necessità di apportare alcuna modifica. Infatti, il compilatore TypeScript (chiamato anche transpiler) traduce un'applicazione TypeScript in un'applicazione JavaScript che può essere eseguita su qualsiasi engine.

TypeScript è stato pensato per facilitare lo sviluppo di grandi applicazioni.

4.2.1.1 I motivi della nascita di TypeScript

Chi è abituato a programmare in Java, C#, C++ o qualsiasi altro linguaggio orientato agli oggetti, potrebbe trovare difficoltà al passaggio ad un linguaggio come JavaScript che non effettua controlli statici sui tipi di dato e converte implicitamente i tipi. Se da un lato questa flessibilità è stata uno dei motivi principali dell'enorme successo di JavaScript, dall'altro lato la mancanza di controlli sui tipi all'atto della compilazione può rendere più difficile scovare errori, specialmente quando si sviluppano applicazioni di grosse dimensioni e molto complesse.

Microsoft, con lo sviluppo di TypeScript, ha superato questo limite. TypeScript quindi è dotato del controllo statico dei tipi e offre altre funzionalità che vengono in aiuto specialmente quando si sviluppano applicazioni complesse.

4.2.1.2 Caratteristiche del linguaggio

Come già accennato in precedenza, TypeScript estende JavaScript e le principali funzionalità aggiunte sono:

- Annotazione dei tipi e controllo dei tipi compile-time
- Type inference
- Type erasure
- Interfacce
- Tipo Enum

TypeScript aggiunge anche il supporto per alcune funzionalità proposte nello standard ECMAScript 6:

- Classi (con integrazione del supporto per le annotazioni dei tipi)
- Moduli
- Sintassi abbreviata per definire le funzioni anonime tramite l'operatore '=>'

4.2.1.3 Annotazione tipizzata

Per abilitare il controllo dei tipi durante la fase di compilazione TypeScript fornisce un sistema di annotazione dei tipi. Questa funzionalità è opzionale e può essere tralasciata se si vuole continuare ad utilizzare il meccanismo di tipi dinamico di JavaScript.

```
function add(left: number, right: number): number {
     return left + right;
}
```

Per i tipi primitivi sono disponibili le annotazioni *number, boolean* e *string*. Le strutture invece sono di tipo *any*.

Il compilatore TypeScript fa uso della *type inference* per desumere un tipo quando questo non è esplicitamente fornito. Se nel metodo *add* dell'esempio precedente non fosse stato indicato alcun tipo di ritorno, il compilatore avrebbe dedotto ugualmente il tipo di ritorno basandosi sul fatto che i due parametri in ingresso sono di tipo *number* e che il risultato della somma di due

number è sempre un *number*. Comunque, dichiarare il tipo di ritorno permette al compilatore di verificare la correttezza.

Nel caso in cui non può essere dedotto alcun tipo, allora viene assegnato il tipo *any*. Un valore al quale è stato assegnato il tipo *any* supporta le stesse operazioni di un valore in JavaScript.

4.2.1.4 Interfacce

Durante la compilazione di un file TypeScript c'è la possibilità di creare un *declaration file* che rappresenta un'interfaccia per i componenti nel JavaScript compilato. Durante questa fase il compilatore elimina i body di tutte le funzioni e i metodi conservando solo le firme dei tipi che sono esportati. Il file generato da questo processo può essere utilizzato per descrivere i tipi TypeScript esportati di una libreria JavaScript o di un modulo, nel momento in cui uno sviluppatore di terze parti ne farà utilizzo tramite TypeScript.

4.2.2 Architettura di un'applicazione Angular

Angular permette di sviluppare applicazioni estremamente modulari. Infatti, un'applicazione del genere è costituita da diversi moduli, che prendono il nome di NgModule: il modulo principale viene chiamato solitamente AppModule ed è il responsabile ad avviare l'applicazione.

Ogni NgModule possiede due elementi fondamentali: un Template ed un Component. Il primo definisce come una pagina appare graficamente all'utente, mentre il secondo si occupa dell'interazione della grafica con la logica applicativa.

La comunicazione con il web server viene gestita utilizzando i Service, che sono un'altra componente fondamentale delle applicazioni Angular. I Service si incaricano di effettuare le chiamate REST necessarie ad accedere ai database.

4.2.2.1 Moduli

Tutte le applicazioni Angular sono dotate di almeno un modulo, l'AppModule. Questo potrebbe anche essere l'unico modulo dell'applicazione, nel caso in cui questa sia di dimensioni limitate. Man mano che la complessità di un'applicazione aumenta conviene creare moduli diversi, assegnando una specifica funzionalità dell'applicazione ad ognuno di essi.

Un NgModule è una classe alla quale è associato il decoratore '@NgModule'. I decoratori sono delle funzioni che aggiungono metadata ad una classe per indicare come dovrebbe funzionare la classe stessa. Il decoratore '@NgModule' riceve un solo oggetto metadata in ingresso con delle proprietà che descrivono il modulo. Le proprietà principali sono:

- Declarations: l'elenco delle *view classes* che fanno parte del modulo. Angular distingue tre tipi di view classes: i componenti, le direttive e le pipes.
- Exports: sottoinsieme delle declarations che sono visibili e utilizzabili all'interno dei componenti che fanno parte di altri moduli.

- Imports: elenco dei moduli che esportano delle classi utilizzate all'interno del modulo corrente.
- Providers: elenco dei servizi generati all'interno del modulo e che possono essere acceduti da tutte le parti dell'applicazione.
- Bootstrap: indica la view principale dell'applicazione, che ingloba tutte le altre view. Questa proprietà dovrebbe essere settata solo all'interno del decoratore dell'AppModule.

Un esempio di AppModule è quello fornito dalla documentazione ufficiale di Angular:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
      imports: [ BrowserModule ],
      providers: [ Logger ],
      declarations: [ AppComponent ],
      exports: [ AppComponent ],
      bootstrap: [ AppComponent ]
})
export class AppModule { }
```

In questo esempio non ci sarebbe bisogno di esportare l'AppComponent: ciò serve solo a mostrare come utilizzare l'array degli Exports. Infatti, l'AppModule non esporta nulla dato che gli altri componenti non hanno necessità di importare il modulo principale.

4.2.2.2 Componenti

I componenti si occupano di gestire parti dello schermo che prendono il nome di view. La logica applicativa di un componente, cioè il modo in cui esso si relaziona alla view ad esso associata, è descritta all'interno di una classe. La comunicazione tra la view e il component avviene utilizzando un API di metodi e proprietà.

Un esempio di componente è il seguente, fornito dalla documentazione ufficiale di Angular:

```
export class HeroListComponent implements OnInit {
      heroes: Hero[];
      selectedHero: Hero;
      constructor(private service: HeroService) { }
      ngOnInit() {
            this.heroes = this.service.getHeroes();
      }
      selectHero(hero: Hero) {
            this.selectedHero = hero;
```

```
        }
    }
```

Il componente mostrato in questo esempio possiede una proprietà 'heroes' che viene popolata con i dati acquisiti da un service. Il metodo 'selectHero' viene invocato quando l'utente clicca su uno degli hero della lista e aggiorna di conseguenza la proprietà 'selectedHero'.

Angular gestisce il ciclo di vita dei componenti creandoli, aggiornandoli o distruggendoli in base a come l'utente naviga all'interno dell'applicazione. Tramite i metodi quali 'ngOnInit' gli sviluppatori possono specificare il comportamento di un componente nelle varie fasi del suo ciclo di vita.

4.2.2.3 Template

I template sono costituiti da codice HTML che indica ad Angular come effettuare il render di un componente. I template possono combinare i tag tipici dell'HTML con le sintassi proprie di Angular.

Un esempio di template è il seguente, fornito dalla documentazione ufficiale di Angular ed associato al componente HeroListComponent introdotto nel paragrafo precedente:

```
<h2>Hero List</h2>
<p><i>Pick a hero from the list</i></p>
<ul>
      <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
            {{hero.name}}
      </li>
</ul>
<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero">
</app-hero-detail>
```

Codice come **ngFor, {{hero.name}}, (click), [hero], <app-hero-detail>* deriva dalla sintassi Angular. In particolare, il tag *<app-hero-detail>* è associato ad un nuovo componente Angular. Questo componente (il cui codice non viene mostrato) serve a presentare i dettagli di uno specifico hero selezionato dall'utente ed è figlio del componente HeroListComponent introdotto nel paragrafo precedente.

4.2.2.4 Metadata

Come già accennato in precedenza, i metadata indicano ad Angular come trattare una classe. Tornando all'esempio relativo al componente HeroListComponent non c'è alcun decoratore ad indicare che si tratta di un componente; quindi HeroListComponent è solo una classe come tutte le altre. Per renderla un componente bisogna aggiungere il decoratore *@Component* che riceve un oggetto metadata con tre proprietà:

- Selector: indica ad Angular che bisogna creare un'istanza del componente ogni qual volta nel codice HTML del componente padre viene trovato un tag con il valore uguale a quello espresso dal selettore.
- TemplateUrl: indirizzo relativo al modulo all'interno del quale si trova il componente che indica dove trovare il template HTML del componente stesso.
- Providers: elenco dei service richiesti dal componente. Questo non è l'unico modo per indicare ad Angular che un componente necessità di servizi.

Un esempio di metadata che indica di trattare una classe come un componente è il seguente:

```
@Component({
      selector: 'app-hero-list',
      templateUrl: './hero-list.component.html',
      providers: [ HeroService ]
})
export class HeroListComponent implements OnInit { /* . . . */ }
```

L'insieme di template, metadata e componente costituisce una view.

4.2.2.5 Data binding

Questa funzionalità permette di relazionare parti di un template con parti di un componente. Angular offre quattro forme di sintassi per effettuare data binding. Ad ogni forma è associata una direzione, che potrebbe essere dal componente al DOM, dal DOM al componente o verso entrambe le direzioni. Le quattro forme di data binding sono:

- Interpolazione: visualizza il valore di una proprietà del componente all'interno di un elemento HTML. Un esempio potrebbe essere *
{{hero.name}}
*.
- Property binding: permette di settare una proprietà di un elemento con il valore di una proprietà del componente. Un esempio potrebbe essere *<app-hero-detail [hero]="selectedHero"></app-hero-detail>*, in cui la proprietà hero del componente figlio HeroDetailComponent è settata con il valore della proprietà *selectedHero* del componente padre.
- Event binding: permette di reagire ad un qualsiasi evento del DOM, come gli input utente. Un esempio potrebbe essere *<li (click)="handleClick()">
*, in cui all'evento click è associata la chiama al metodo *handleClick.*
- Two-way binding: unisce property binding ed event binding in una singola notazione tramite la direttiva *ngModel*. Infatti, da un lato il valore di una proprietà viene passato dal component ad un elemento di input come nel property binding, mentre dall'altro lato il cambiamento dell'input utente viene passato al componente modificandone il valore della proprietà.

Quindi il data binding è fondamentale sia nella comunicazione tra un template e il componente ad esso associato, sia tra un componente padre e un componente figlio.

4.2.2.6 Direttive

I template in Angular sono dinamici. Nel momento in cui ne viene fatto il rendering, Angular modifica il DOM seguendo le istruzioni fornite dalle direttive.

Una direttiva è una classe alla quale è associata un decoratore *@Directive*. Un componente può essere considerato come una direttiva con un template. Infatti, il decoratore *@Component* è un decoratore *@Directive* arricchito da caratteristiche orientate al template.

Angular propone altri due tipi di direttive: quelle strutturali e quelle attributi. Entrambe si mostrano all'interno di un tag allo stesso modo in cui sono utilizzati gli attributi degli elementi HTML.

Le direttive strutturali modificano il layout rimpiazzando, aggiungendo o eliminando elementi nel DOM. **ngFor* e **ngIf* sono le due direttive strutturali principali. Un esempio di utilizzo della direttiva **ngFor* è *<li *ngFor="let hero of heroes">
*, tramite la quale Angular aggiunge un elemento *<li>* per ogni *hero* appartenente alla lista *heroes*. Un esempio di utilizzo della direttiva **ngIf* è *<app-hero-detail *ngIf="selectedHero"></app-hero-detail>*, tramite la quale il componente *HeroDetailComponent* viene mostrato solo se *selectedHero* esiste.

Le direttive attributi servono a modificare l'aspetto o il comportamento di un elemento. Un esempio di direttiva attributo è la direttiva *ngModel*.

4.2.2.7 Service

È fondamentale premettere che Angular non ha una definizione specifica di service e non predispone alcuna classe base per i service. Quindi, tutto potrebbe essere un service. Si può guardare ad un service come ad una classe con una funzionalità ben precisa.

Esempi di service sono quelle classi che offrono un meccanismo di log, che permettono di configurare l'applicazione o che permettono di inviare richieste http ad un web server. I service sono fondamentali per i componenti Angular. Un component, infatti, non dovrebbe occuparsi di scaricare dati da un server, o effettuare dei log direttamente in console. Questi compiti dovrebbero essere sempre delegati ai service.

Qui di seguito viene mostrato un esempio di classe service che effettua il log sulla console del browser:

```
export class Logger {
      log(msg: any) {
            console.log(msg);
      }
      error(msg: any) {
            console.error(msg);
      }
      warn(msg: any) {
            console.warn(msg);
```

```
        }
    }
```

Angular non obbliga ad utilizzare i service, ma tramite il meccanismo del dependency injection rende molto semplice l'utilizzo dei service all'interno dei componenti.

4.2.2.8 Dependency injection

Il dependency injection permette di creare una nuova istanza di una classe e fornirle tutte le dipendenze di cui necessita. Solitamente queste dipendenze sono rappresentate da service. Con questo meccanismo, quando Angular crea un nuovo componente può fornirgli tutti i service da esso richiesti.

Quando un componente ha bisogno di utilizzare un service, lo indica tra i parametri del suo costruttore. Il tipo del parametro indica il tipo particolare di service che Angular deve fornire al componente. Per fare ciò è fondamentale il ruolo ricoperto dall'Injector. Questo funge da contenitore per tutte le istanze di service create in precedenza. Nel momento in cui Angular deve creare un nuovo componente chiede all'injector di fornirgli tutti i service che sono elencati nel costruttore del componente. Se per qualcuno di questi service non è mai stata creata un'istanza, l'injector la crea. Quando tutti i service richiesti all'injector vengono risolti e ritornati, allora Angular può chiamare il costruttore del componente, passandogli come argomenti i service richiesti.

Per poter permette all'injector di creare un service, bisogna prima registrare un provider del service stesso. I provider possono essere registrati sia nei moduli che nei componenti. In generale, conviene aggiungere i provider all'interno dell'AppModule: in questo modo tutta l'applicazione potrà accedere alla stessa istanza del service. Nel caso in cui il provider venga aggiunto direttamente nel decoratore del component, ogni qual volta viene creata una nuova istanza del component verrà creata anche una nuova istanza del service.

La "Figura 8" mostra l'architettura tipica di un'applicazione Angular.

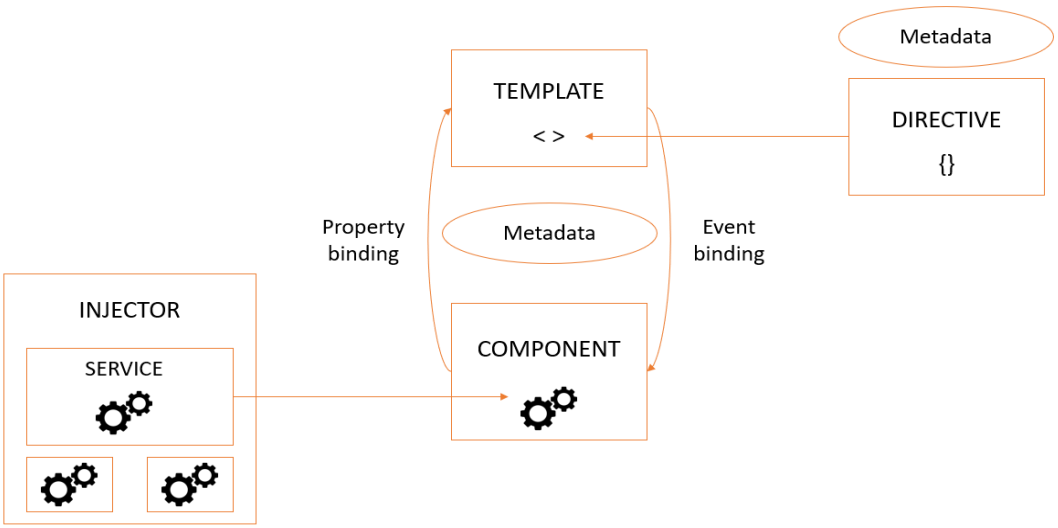


***Figura 8**: Architettura Angular*

4.2.3 Vantaggi

Dopo aver introdotto e descritto Angular e Typescript è più facile focalizzarsi su quelli che sono i vantaggi offerti dal framework e capire perché è stato scelto nello sviluppo dell'applicazione oggetto di questo elaborato di tesi:

- Manutenibilità: una tipica applicazione Angular è sviluppata in più moduli. Tramite essi è più facile organizzare le funzionalità offerte dall'applicazione suddividendole in blocchi di codice riusabili. Poiché le applicazioni enterprise possono raggiungere dimensioni davvero elevate, Angular offre un grosso aiuto nel mantenere il codice organizzato. La manutenibilità del codice Angular è dovuta anche alla netta separazione dei ruoli tra component e template, che rende anche il codice molto più leggibile nei confronti per esempio di un'applicazione con analoghe funzionalità, ma sviluppata interamente in JQuery.
  Un altro aspetto di Angular che rende le applicazioni più mantenibili è l'utilizzo della dependency injection: ogni component è disaccoppiato dalle proprie dipendenze, le quali sono eseguite parallelamente ai componenti. Quindi, le classi utilizzano le loro dipendenze ottenendole da una sorgente esterna, cioè l'injector. Quanto detto permette anche di ottenere prestazioni migliori rispetto per esempio all'utilizzo di AngularJS.
- Efficienza: l'introduzione della compilazione Ahead-of-Time (AOT) permette di velocizzare il rendering da parte del browser. Mentre con la normale compilazione Just-in-time (JIT) l'applicazione viene compilata dal browser in runtime, con la compilazione AOT il browser scarica una versione dell'applicazione già precompilata. In questo modo l'applicazione può essere renderizzata immediatamente visto che la compilazione è già avvenuta. L'utilizzo della compilazione AOT porta anche altri due vantaggi relativi all'efficienza. Il primo è che non serve effettuare chiamate ajax per ottenere i template HTML esterni e gli style sheet CSS, poiché il compilatore li inserisce direttamente all'interno dell'applicazione JavaScript compilata. Il secondo è che non c'è bisogno di scaricare il compilatore Angular se l'applicazione è già stata compilata e ciò permette di ridurre il payload dell'applicazione.
- Testabilità: il fatto di avere componenti indipendenti tra di loro semplifica molto la realizzazione di unit test, tramite i quali testare ogni piccola parte dell'applicazione.
- Gestione degli errori: l'utilizzo di TypeScript permette di scovare eventuali errori già durante la compilazione, a differenza di tutte le altre applicazioni scritte in JavaScript.
- Supporto di Google a lungo termine: questo è un altro grande vantaggio del framework, dato che la stessa Google ha annunciato un supporto a lungo termine per Angular.

4.3 Back-end: Asp.NET Web API 2.0

Il protocollo HTTP non serve solo a fornire web page. Infatti, può essere utilizzato anche come uno strumento potente per compilare API che mettono a disposizione servizi e dati. Poiché al giorno d'oggi la maggior parte delle piattaforme esistenti dispone di librerie HTTP, diventa semplice capire quant'è ampio il raggio d'azione dei servizi HTTP, che possono comunicare con browser desktop, dispositivi mobili o applicazioni desktop tradizionali.

Asp.NET Web API si basa sulla piattaforma Asp.NET ed è un framework open source che permette di costruire applicazioni RESTful in modo molto semplice. Il funzionamento è molto

simile a quello delle applicazioni web Asp.NET MVC, con la differenza che vengono inviati dei dati e non delle view HTML in risposta alle richieste HTTP.

4.3.1 Perché Asp.NET

Per lo sviluppo del back-end è stata scelta la piattaforma Asp.NET principalmente per via del suo ambiente unificato fornito da Microsoft. Non dipendendo da venditori di terze parti, la qualità mantenuta è sempre elevata. Inoltre la piattaforma Asp.NET offre un'alta scalabilità orizzontale e permette di effettuare il deploy di una web application in modo molto semplice e veloce. Infine, l'IDE standard offerto da Microsoft, Visual Studio, possiede tutto ciò che serve ad un developer per lo sviluppo, il debug, il build e il deploy senza la necessità di installare altri tool.

Web API 2.0 è lo strumento ideale per creare applicazioni RESTful all'interno della piattaforma Asp.NET, tramite il quale esporre servizi e dati a tipologia diverse di device. Il paragrafo successivo elenca le caratteristiche principali offerte da questo framework.

4.3.2 Caratteristiche

Le caratteristiche principali offerte dal framework sono:

- Supporta le Action basate sulla convenzione CRUD dato che fa utilizzo dei verbi HTTP GET, POST, PUT e DELETE. Quindi è un'ottima piattaforma per la realizzazione di servizi RESTful.
- Mappa automaticamente i verbi HTTP ai nomi dei metodi.
- Offre un supporto nativo per formattare le risposte HTTP in formato JSON o XML. Non è richiesta alcuna conversione esplicita per gli oggetti ritornati a seguito di richieste HTTP poiché è il framework stesso che si occupa di trasformarli in XML o JSON prima di ritornarli al client.
- Può essere ospitato all'interno dell'applicazione stessa (self-hosted), in IIS o in un qualsiasi altro web server che supporti .NET 4.0+.
- Può ricevere o inviare contenuti non object-oriented, come file pdf o immagini.
- Le risorse sono mappate direttamente sui controller. Solitamente gli sviluppatori creano un controller per ognuno dei tipi di dato principali.
- Utilizza il motore di routing della piattaforma .NET per mappare gli URL ai controller.
- Facendo parte della famiglia Asp.NET supporta in modo nativo funzionalità quali l'Entity Framework e la gestione dell'autenticazione Windows.
- Costruisce automaticamente la documentazione relativa alle API includendo anche degli esempi.

La "Figura 9" mostra un tipico contesto di utilizzo delle Web API.

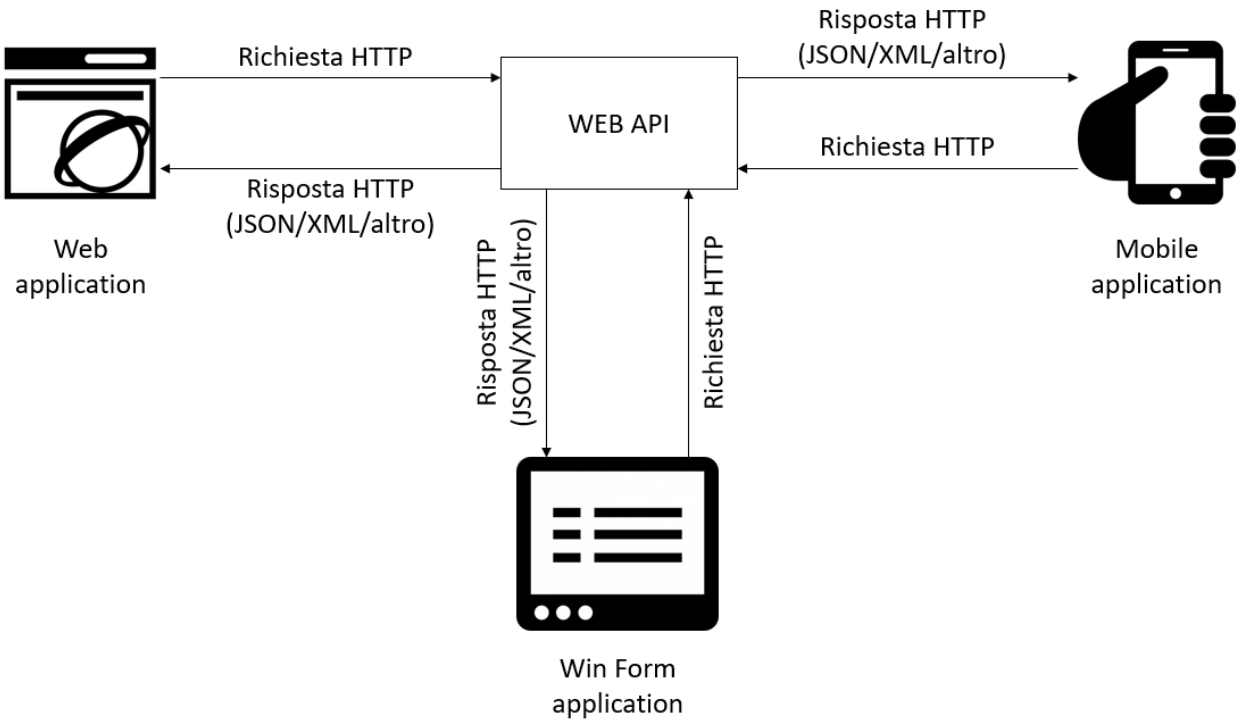


***Figura 9**: Web API*

4.3.3 Controller

Il controller Web API serve a gestire richieste HTTP provenienti dai client generando risposte HTTP che vengono rispedite al chiamante. Ogni controller deve estendere la classe *System.Web.Http.ApiController* e il nome della classe controller deve terminare con "Controller". I metodi pubblici definiti all'interno di un controller prendono il nome di Action method.

Un esempio di controller è quello che viene generato automaticamente da Visual Studio quando si crea un nuovo progetto Web API:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace WebApplication1.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        public string Get(int id)
        {
```

```
            return "value";
        }

        // POST api/values
        public void Post([FromBody]string value)
        {
        }

        // PUT api/values/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE api/values/5
        public void Delete(int id)
        {
        }
    }
}
```

Quando sopraggiunge una nuova richiesta HTTP, l'URL viene utilizzato per localizzare il controller responsabile a gestire la richiesta, mentre il verbo HTTP della richiesta viene utilizzato per chiamare uno dei metodi del controller. Se la richiesta HTTP è una POST, allora verrà chiamato il metodo *Post* del controller; se non ci fosse stato il metodo *Post* sarebbe stato chiamato quel metodo il cui nome inizia con "Post".

Non si è obbligati a dare ad un metodo del controller un nome che inizi per uno dei verbi http: in questi casi si può utilizzare un attributo opportuno per associare al metodo un verbo HTTP, come per esempio:

```
// GET api/values
[HttpGet]
public IEnumerable<string> Values()
{
       return new string[] { "value1", "value2" };
}
```

4.3.4 Routing

Web API offre due tipi di routing: convention-based routing e attribute routing.

Nel primo caso, vengono utilizzati i route template per determinare quale controller e quale action method eseguire. Alla creazione di un progetto Web API viene generata una classe *WebApiConfig* con una route di default.

```
public static class WebApiConfig
{
       public static void Register(HttpConfiguration config)
        {
            // Route dell'API Web
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
```

```
            );
        }
}
```

La *config.Routes* è una tabella di routing all'interno della quale è stata aggiunta una route chiamata *DefaultApi*. Nel caso in cui dovesse sopraggiungere una GET all'URL http://localhost:1234/api/values, allora questa andrà in match con il route template della *DefaultApi* /api/{controller}/{id} in cui il parametro *id* è stato specificato come opzionale. Il valore di {controller} sarà infatti *ValuesController* e verrà chiamato il suo metodo *Get*. Nel caso in cui il framework non trova nessun match tra l'URL di una richiesta ricevuta e le route configurate verrà generata una risposta HTTP di tipo Not Found (404).

Web api offre anche la possibilità di configurare molteplici route all'interno della tabella *config.Routes.*

L'attribute routing è stato introdotto a partire dalle Web API 2. Per poter utilizzare questa funzionalità, bisogna abilitarla all'interno della classe *WebApiConfig* chiamando il metodo *config.MapHttpAttributeRoutes.* Per definire delle route seguendo questo tipo di convenzione si utilizza l'attributo *[Route()]* applicandolo ad un qualsiasi controller o ad un qualsiasi action method. Quindi, se per esempio l'attributo viene applicato ad un certo action method, questo viene invocato quando sopraggiunge una richiesta HTTP con URL coincidente a quello espresso dall'attributo.

4.3.5 Binding dei parametri

Gli action method possono richiedere uno o più parametri di tipi differenti e possono essere sia di tipo primitivo che di tipo complesso. Il binding dei parametri dipende dal tipo di parametro stesso: il comportamento di default è che se il parametro è di tipo primitivo o di un qualsiasi tipo convertibile da stringa allora il valore viene settato dalla query string, mentre se è di tipo complesso allora il framework prova ad ottenere il valore dal body della richiesta HTTP.

Per modificare questo comportamento di default è possibile utilizzare gli attributi *[FromUri]* oppure *[FromBody]*: il primo può essere utilizzato per indicare al framework di ottenere il valore di un tipo complesso dalla query string, mentre il secondo per indicare di ottenere il valore di un tipo primitivo dal body della richiesta. Un esempio di utilizzo di questi attributi è il seguente:

```
public Student Get([FromUri] Student stud) { }
```

4.3.6 Tipo di ritorno di un action method

Un action method può ritornare uno tra i seguenti quattro tipi: void, tipo primitivo o tipo complesso, *HttpResponseMessage*, *IHttpActionResult*.

Un action method potrebbe non avere necessità di ritornare qualcosa. In questo caso il tipo di ritorno è void. Per esempio, un action method che ha lo scopo di eliminare una risorsa non ha bisogno di ritornare nulla al chiamante. Settando il tipo di ritorno a void, la risposta che verrà inviata al client sarà di tipo 204 (No content).

In secondo luogo, un action method può ritornare un tipo primitivo o complesso alla pari di qualsiasi altro metodo.

La terza possibilità è quella di poter ritornare un oggetto di tipo *HttpResponseMessage*. In realtà un controller ritorna sempre un oggetto di questo tipo all'infrastrutture host, anche quando il tipo di ritorno scelto per un action method è void, primitivo o complesso. Il vantaggio di ritornare un *HttpResponseMessage* direttamente da un action method è quello di poter configurare la risposta a proprio piacimento: è possibile settare liberamente il codice di stato della risposta, il contenuto o un messaggio di errore. Un esempio del genere è il seguente, in cui se non viene trovato alcuno studente il client riceverà una risposta HTTP di tipo Not found:

```
public HttpResponseMessage Get(int id)
{
	Student stud = GetStudentFromDB(id);
	if (stud == null)
	{
		return Request.CreateResponse(HttpStatusCode.NotFound, id);
	}
	 return Request.CreateResponse(HttpStatusCode.OK, stud);
}
```

La quarta e ultima possibilità è quella di ritornare un oggetto di una classe che implementi l'interfaccia *IHttpActionResult*. In questo caso è possibile definirsi da sé una classe che implementa questa interfaccia o invocare uno dei metodi della classe *ApiController* che hanno come tipo di ritorno un oggetto che implementa questa interfaccia. Un esempio è il seguente:

```
public IHttpActionResult Get(int id)
{
	Student stud = GetStudentFromDB(id);
	if (stud == null)
	{
		return NotFound();
	}
	return Ok(stud);
}
```

Nell'esempio soprastante i metodi *NotFound* e *Ok* ritornano un oggetto che implementa l'interfaccia *IHttpActionResult*.

4.3.7 Formato dei dati per le richieste/risposte

Nei messaggi HTTP il media type indica il formato dei dati specificandone il tipo e il sottotipo, come per esempio text/html, text/xml, application/json, image/jpeg e così via.

Nelle richieste HTTP l'header *Accept* indica che tipo di formato si aspetta il client all'interno della risposta HTTP, mentre l'header *Content-Type* indica il formato dei dati inviati nel body della richiesta. Se per esempio il client si aspetta di ricevere una risposta in formato Json, assegnerà all'header *Accept* della richiesta HTTP il valore application/json.

Il framework è dotato di un supporto nativo per i formati Json, xml, Bson e form-urlencoded. Ciò vuol dire che i dati delle richieste e delle risposte sono convertiti automaticamente in questi formati.

Si prenda in esame il seguente esempio:

```
public class Student
{
	public int Id { get; set; }
	public string Name { get; set; }
}

public class StudentController : ApiController
{
	public Student Post(Student student)
	{
		// save student into db
		var insertedStudent = SaveStudent(student);
		return insertedStudent;
	}
 }
```

Il metodo *Post* dell'esempio soprastante riceve in ingresso un oggetto di tipo Student e ritorna un oggetto dello stesso tipo. Il body della richiesta HTTP potrebbe essere sia in formato Json che in formato xml: in base al valore dell'header *Content-Type* il framework sa se utilizzare un *JsonMediaTypeFormatter* o un *XmlMediaTypeFormatter* per convertire i dati ricevuti in un oggetto Studente. In seguito, in base al valore dell'header *Accept*, l'oggetto *insertedStudent* verrà serializzato in formato Json o xml.

4.3.8 Filtri

I filtri possono essere utilizzati per introdurre funzionalità extra prima o dopo l'esecuzione degli action method. Tramite i filtri è possibile per esempio implementare meccanismi di logging, gestire le eccezioni o autenticare e autorizzare gli utenti.

I filtri sono degli attributi e possono essere applicati sia ai controller che agli action method. Ogni qual volta si voglia utilizzare un filtro è necessario implementare l'interfaccia *IFilter* del namespace *System.Web.Http.Filters*. Per semplificare il lavoro degli sviluppatori, all'interno di questo namespace sono presenti delle interfacce e delle classi più specifiche dell'interfaccia *IFilter* che possono essere utilizzare per creare filtri con determinati scopi.

Quando per realizzare un filtro si sceglie di utilizzare una delle interfacce presenti nel namespace allora bisogna implementare tutti i metodi dichiarati nell'interfaccia scelta. Se invece si sceglie una delle classi presenti nel namespace, allora basta solamente effettuare l'override di quei metodi per i quali si vogliono aggiungere funzionalità extra.

4.4 Persistenza dei dati: SQL Server

Microsoft SQL Server è un database management system (DBMS) di tipo relazionale prodotto da Microsoft. Essendo un database server, la sua funzione è quella di salvare e recuperare i dati richiesti da applicazioni che possono essere in esecuzione sullo stesso computer in cui è eseguito il server o su computer differenti connessi ad esso tramite una rete (inclusa anche Internet). Inizialmente era stato pensato per basi di dati di medio/piccole dimensioni. Dagli anni 2000 in poi è stato utilizzato anche per la gestione di database di grosse dimensioni.

4.4.1 Perché SQL Server

Il vantaggio principale di SQL Server è la sua facilità d'uso. SQL Server porta con sé un insieme di tool, quali SQL Server Profiler o SQL Server Management Studio, che rendono molto più semplice lo sviluppo e la risoluzione dei problemi per un progetto SQL permettendo di risparmiare molto tempo.

Essendo uno dei prodotti più diffusi e utilizzati nell'ambito dei database relazionali, il supporto online è molto vasto così come la documentazione. In questo modo è più semplice superare qualsiasi tipo di problema si incontri durante la fase di sviluppo.

4.4.2 Architettura

Per comunicare con un SQL Server si utilizza il Tabular Data Stream (TDS), un formato definito da Microsoft. Quindi TDS è un protocollo a livello applicativo che permette di scambiare dati tra un client e un server database. I pacchetti TDS possono essere trasportati utilizzando vari protocolli di trasporto, quali TCP/IP, named pipes e shared memory. Inoltre, le API di SQL Server sono esposte anche su servizi web.

4.4.3 Salvataggio dei dati

I dati sono archiviati in un database, cioè un insieme di tabelle con colonne tipizzate. I tipi supportati da SQL Server sono sia tipi primitivi, quali Char (incluse stringhe di caratteri), Varchar (stringhe di caratteri con lunghezza variabile), Decimal, Integer, Float, Binary e Text, sia tipi complessi definiti dall'utente. All'interno di un database possono anche essere contenuti indici, vincoli, view e store procedure. La dimensione massima di un database SQL Server è di un exabyte e il numero massimo di oggetti che può contenere è 2^{31}. I dati di un database sono salvati in file dati primari con estensione $.mdf$. SQL Server include anche dei file dati secondari, con estensione $.ndf$, che permettono di spezzettare i dati di un singolo database in più file, e volendo anche in più file system. I file di log invece sono caratterizzati dall'estensione $.ldf$.

L'unità base per le operazioni di I/O in SQL Server è la pagina: ognuna di esse è grande 8 KB. Queste pagine sono numerate in modo sequenziale e costituiscono lo spazio allocato per il database. Ogni pagina ha un header di 96 byte che contiene dei metadati sulla pagina stessa, come il numero della pagina, il tipo della pagina, lo spazio libero in essa e l'ID dell'oggetto che possiede la pagina. Il tipo di pagina indica che tipo di dati contiene la pagina: esempi di tipi di dato sono gli indici, i dati salvati nel database, le allocation map (informano come le pagine vengono allocate alle tabelle e agli indici), le change map (informano sui cambiamentti effettuati su altre pagine a seguito dell'ultimo backup). Nonostante si sia parlato di pagine, lo spazio di allocazione viene visto in termini di extent, che sono gruppi di otto pagine. Un oggetto del database può occupare tutte le otto pagine di un extent oppure può condividere un extent con altri oggetti (al massimo 7). Le righe di un database non possono essere più grandi della dimensione di una pagina. Nel caso in cui una o più colonne di una riga siano di tipo varchar o binario e i dati vadano oltre gli 8 KB, allora questi dati sono espansi su una pagina o una sequenza di pagine.

Le righe di una tabella sono suddivise in una serie di partizioni (numerate da 1 a n) durante il salvataggio fisico. La dimensione delle partizioni è definita dall'utente. Nonostante in una situazione di default tutte le righe facciano parte di una singola partizione, la suddivisione in più partizioni serve a distribuire il database su un cluster di computer.

4.4.3.1 Gestione del buffer

Il buffer serve a rendere minimo il numero di operazioni di I/O sul disco. L'insieme delle pagine bufferizzate costituisce la buffer cache. Il numero di pagine che possono essere cachate dipende dalla quantità di memoria messa a disposizione per SQL Server. Ogni qual volta si accede ad una pagina in lettura e scrittura questa viene copiata nella cache. In questo modo i successivi accessi alla pagina, sia in lettura che in scrittura verranno eseguiti utilizzando la cache. La versione della pagina aggiornata viene copiata sul disco solo quando la sua versione in cache non è stata acceduta per un certo periodo di tempo. Assieme alla pagina, viene salvato su disco anche il suo checksum. Quando si rilegge la pagina, si ricalcola il suo checksum e lo si confronta con quello salvato precedentemente per controllare se la pagina sia danneggiata o meno.

4.4.3.2 Gestione della concorrenza

SQL Server permette a più client di accedere contemporaneamente allo stesso database. Ciò implica la necessità di gestire gli accessi concorrenti ai dati condivisi in modo da evitare che i dati vengano corrotti quando più client stanno aggiornando lo stesso dato o un client tenta di leggere un dato che è in fase di aggiornamento a seguito della richiesta di un altro client. Microsoft ha pensato a due possibili modi di gestire la concorrenza: quello pessimistico e quello ottimistico.

Nel caso della concorrenza pessimistica vengono utilizzati i lock, che possono essere sia esclusivi che condivisi. Nel primo caso l'accesso al dato condiviso è esclusivo: fintantoché viene posseduto il lock, nessun altro utente può accedere allo stesso dato. Il secondo caso permette a più client di leggere contemporaneamente lo stesso dato: l'acquisizione del lock esclusivo presuppone che tutti i lock condivisi sullo stesso dato siano stati rilasciati. Il livello di granularità con cui possono essere utilizzati i lock è molto vario: un lock potrebbe essere applicato ad un'intera tabella, ad un'intera pagina o ad un'intera riga di una tabella. Il livello di granularità viene deciso dall'amministratore del database. Più fine è il livello di granularità scelto, più utenti potranno accedere contemporaneamente ad una stessa tabella, ma anche più risorse verranno richieste. Questo fattore può impattare notevolmente sulle prestazioni. È per questo che SQL Server offre altri due meccanismi di mutua esclusione, latche e spinlock, caratterizzati da migliori prestazioni, ma da una minore robustezza. SQL Server inoltre controlla l'esecuzione dei thread che acquisiscono i lock in modo da evitare che si creino dei deadlock: quando ciò si verifica uno dei thread che ha causato il deadlock viene terminato e viene effettuato il rollback della transazione ad esso associata.

La concorrenza ottimistica permette di creare una nuova versione di una riga ogni qualvolta essa viene aggiornata (senza sovrascrivere la vecchia versione della riga): la nuova riga viene identificata utilizzando anche l'ID della transazione che l'ha creata. La vecchia versione della

riga non viene eliminata, ma viene spostata nel database di sistema chiamato Tempdb. Quando si aggiorna una riga, tutte le altre richieste vengono eseguita sulla vecchia versione e non vengono bloccate. Se una di queste altre richieste è un update, due versioni della stessa riga vengono salvate nel database, identificate dai rispettivi ID delle transazioni.

4.4.4 Accesso ai dati e programmabilità

I dati salvati all'interno di un database sono acceduti principalmente tramite l'utilizzo di query. Il linguaggio utilizzato per le query è il T-SQL (Transact-SQL) che è un dialetto di SQL introdotto da Microsoft. Le query sono processate da un query processor, che stabilisce la sequenza delle azioni utili a ottenere i dati richiesti (il cosiddetto query plan). Ad una query potrebbero essere associati diversi query plan: in questi casi SQL Server sceglie quel piano che permette di accedere ai dati nel minor tempo possibile. La valutazione di quale query plan adottare è a carico dello stesso query processor.

SQL Server fa utilizzo di un ottimizzatore di query per ottimizzare il costo delle query, in termini di risorse necessarie per accedere ai dati. Per compiere il proprio lavoro, l'ottimizzatore delle query prende in esame lo schema del database, le statistiche su di esso e il carico del sistema in quel momento. Di conseguenza decide la sequenza con cui accedere alle tabelle indicate nella query, la sequenza con cui eseguire le operazioni e il metodo da utilizzare per accedere alle tabelle. Inoltre, sceglie se la query debba essere eseguita in modo concorrente o meno. Se da un lato un'esecuzione concorrente potrebbe essere più costosa dal punto di vista del tempo totale di processore, in realtà il fatto di suddividere il compito su più processori potrebbe generare il risultato della query più velocemente. I query plan vengono temporaneamente cachati in modo da poter essere utilizzati per successive invocazioni della stessa query.

In SQL Server è anche possibile definire delle stored procedures, che sono delle query T-SQL parametrizzate salvate all'interno del server stesso. Le stored procedures ricevono dal client valori sotto forma di parametri in input e generano dei risultati sotto forma di parametri in output. Una stored procedure può chiamare altre stored procedure (inclusa sé stessa un numero limitato di volte) o altre funzioni definite. Una differenza tra le stored procedure e le altre query è che è possibili associare ad esse dei nomi, che a runtime vengono risolti nelle vere e proprie query. Tutto ciò permette di ridurre il traffico di rete visto che il codice delle query non deve essere inviato ogni volta dal client.

4.4.4.1 T-SQL

T-SQL è un'estensione del linguaggio SQL, di proprietà Microsoft. Transact-SQL è fondamentale per SQL Server, dato che ogni applicazione che desidera comunicare con un SQL Server invia delle istruzioni scritte in T-SQL.

Le funzionalità extra apportate da T-SQL nei confronti del tradizionale SQL sono:

- Funzioni per il controllo di flusso:
  T-SQL introduce diverse parole chiavi per offrire questa funzionalità, quali *BEGIN, END, BREAK, CONTINUE, GOTO, IF, ELSE, RETURN, WAITFOR, WHILE*. *IF* ed

ELSE permettono di eseguire sequenze di istruzioni in modo condizionale. *BEGIN* ed *END* delimitano un blocco di istruzioni: queste devono essere utilizzate per esempio per indicare un blocco di codice la cui esecuzione dipende da condizioni. Un esempio è il seguente:

```
IF DATEPART(dw, GETDATE()) = 7 OR DATEPART(dw, GETDATE()) = 1
BEGIN
   PRINT 'Oggi è un giorno festivo.'
   PRINT 'Riposati!'
END
ELSE
BEGIN
   PRINT 'Oggi è un giorno feriale.'
   PRINT 'Devi andare a lavorare!'
END
```

WAITFOR serve ad aspettare per un certo periodo di tempo o fino ad un preciso orario del giorno. *RETURN* serve per ritornare immediatamente da una stored procedure o da una funzione. *BREAK* permette di abbandonare un loop cominciato con la keyword *WHILE*, mentre *CONTINUE* passa alla successiva esecuzione di un loop.

- Variabili locali:
  Per dichiarare una variabile locale si usa la keyword *DECLARE*, mentre per settare il valore di una variabile locale si usa la parola chiave *SET*. Un esempio è il seguente:

```
DECLARE @Contatore INT
SET @Contatore = 10
```

- Funzioni per la manipolazione delle espressioni matematiche, delle date e delle stringhe
- Cambiamenti alle istruzioni DELETE e UPDATE:
  T-SQL introduce la possibilità di aggiungere l'opzione *FROM* alle istruzioni DELETE e UPDATE in modo da collegarsi ad altre tabelle tramite JOIN. Un esempio è il seguente:

```
DELETE Utenti
  FROM Utenti as u
  JOIN Utenti_flags as f
    ON u.id=f.id
 WHERE f.name = 'Idle'
```

4.4.4.2 SQL CLR

A partire dalla versione 2005 di SQL Server è stato introdotto il SQL CLR ("Common Language Runtime") che permette di integrare SQL Server con il framework .NET. La peculiarità di SQL Server nei confronti delle altre applicazioni che fanno utilizzo del framework .NET è che esso stesso ospita un framework .NET runtime. Questo vuol dire che le richieste del framework .NET riguardo la gestione della memoria, dei thread e delle risorse vengono prese in carico da SQL Server stesso e non dal sistema operativo Windows sottostante. Con l'introduzione di SQL CLR diventa possibile scrivere stored procedure, trigger e used defined types (UDT) direttamente in uno dei linguaggi .NET, come C# e VB.NET. Tale codice gestito viene compilato in assembly CLI se i vincoli sulla sicurezza dei tipi sono rispettati e a questo punto può essere invocato alla pari di qualsiasi altra procedura.

4.5 Mapping object-relational: Entity Framework

L'Entity Framework è un framework ORM open source per le applicazioni .NET che aiuta gli sviluppatori a concentrarsi sulla progettazione del modello dei dati e non sul database utilizzato per salvarli.

4.5.1 Perché utilizzare un framework ORM come Entity Framework

Nelle applicazioni Asp.NET, prima dell'introduzione dei framework ORM come l'Entity Framework, l'accesso ai dati del database sottostante prevedeva l'utilizzo di codice ADO.NET. Dopo aver aperto la connessione con il database, era necessario creare un DataSet per leggere o scrivere dati nel database, convertendo i dati del DataSet in oggetti .NET o viceversa per applicare le regole di business. Un approccio del genere è tutt'altro che semplice e può condurre facilmente ad errori.

L'utilizzo di un framework per il mapping object-relational evita il bisogno di soffermarsi sulle tabelle e colonne del database in cui i dati sono salvati, perché pone l'attenzione solamente sugli oggetti .NET istanze delle classi di dominio. Quindi il vantaggio principale è quello di essere indipendenti dal database utilizzato: non c'è necessità di scrivere codice per lo specifico database scelto per la persistenza dei dati. Questo fornisce alle applicazioni un'elevata portabilità e permette di cambiare il DBMS senza bisogno di modificare il codice scritto.

Utilizzare un framework per il mapping object-relational vuol dire mantenersi ad un elevato livello di astrazione scrivendo meno codice rispetto alle applicazioni tradizionali e riducendo così il tempo di sviluppo. Infatti, gli sviluppatori possono utilizzare i comandi messi a disposizione dal framework ORM per poter accedere ai dati: è compito del framework tradurre questi comandi in una delle operazioni CRUD (Create, Read, Update, Delete).

Considerando i vari livelli dell'architettura di un'applicazione, il mapping object-relational si posiziona tra le classi di dominio (livello di business) e il database.

Uno dei punti di forza dell'Entity Framework nei confronti degli altri framework ORM è che fornisce un tool per le migrazioni che aggiorna automaticamente lo schema del database a seguito di cambiamenti al modello dei dati, senza perdere i dati già esistenti all'interno del database.

4.5.2 Caratteristiche

Di seguito vengono elencate le caratteristiche principali del framework:

- Modellazione: il framework fa utilizzo dell'Entity Data Model (EDM) basato su entità POCO (Plain Old CLR Object) con proprietà di diversi tipi possibili dotate di getter e setter. Questo modello viene utilizzato ogni qual volta si effettuano delle query al database o si salvano dati all'interno di esso.
- Querying: l'accesso ai dati del database è reso molto semplice se si utilizzano delle query LINQ. La conversione di queste query in un linguaggio specifico per il database viene effettuata dal database provider. L'Entity Framework non esclude comunque la possibilità di effettuare delle query SQL direttamente al database.

- Tracciamento dei cambiamenti: il framework tiene conto di tutte le modifiche effettuate alle istanze delle entità in modo da sottometterle al database.
- Salvataggio: l'inserimento, la cancellazione o la modifica di entità, avviene quando viene chiamato il metodo *SaveChanges()*. Il framework offre anche una versione asincrona del metodo, dal nome *SaveChangesAsync()*.
- Concorrenza: il framework gestisce la concorrenza in modo ottimistico, proteggendo eventuali cambiamenti effettuati da altri utenti dopo che i dati sono stati prelevati dal database.
- Transazioni: EF gestisce automaticamente le transazioni durante le query o il salvataggio dei dati.
- Caching: il framework offre un primo livello di cache in modo che un'eventuale ripetizione di una query ritorni i dati dalla cache e non dal database stesso.
- Convenzioni di default: il framework offre un insieme di regole che configurano automaticamente il modello EF.
- Configurazioni: le convenzioni di default possono essere sovrascritte utilizzando le annotazioni tramite attributi o le FluentAPI.
- Migrazioni: il framework offre diversi comandi che permetto di gestire lo schema del database sottostante.

4.5.3 Architettura

La "Figura 10" mostra l'architettura dell'Entity Framework.

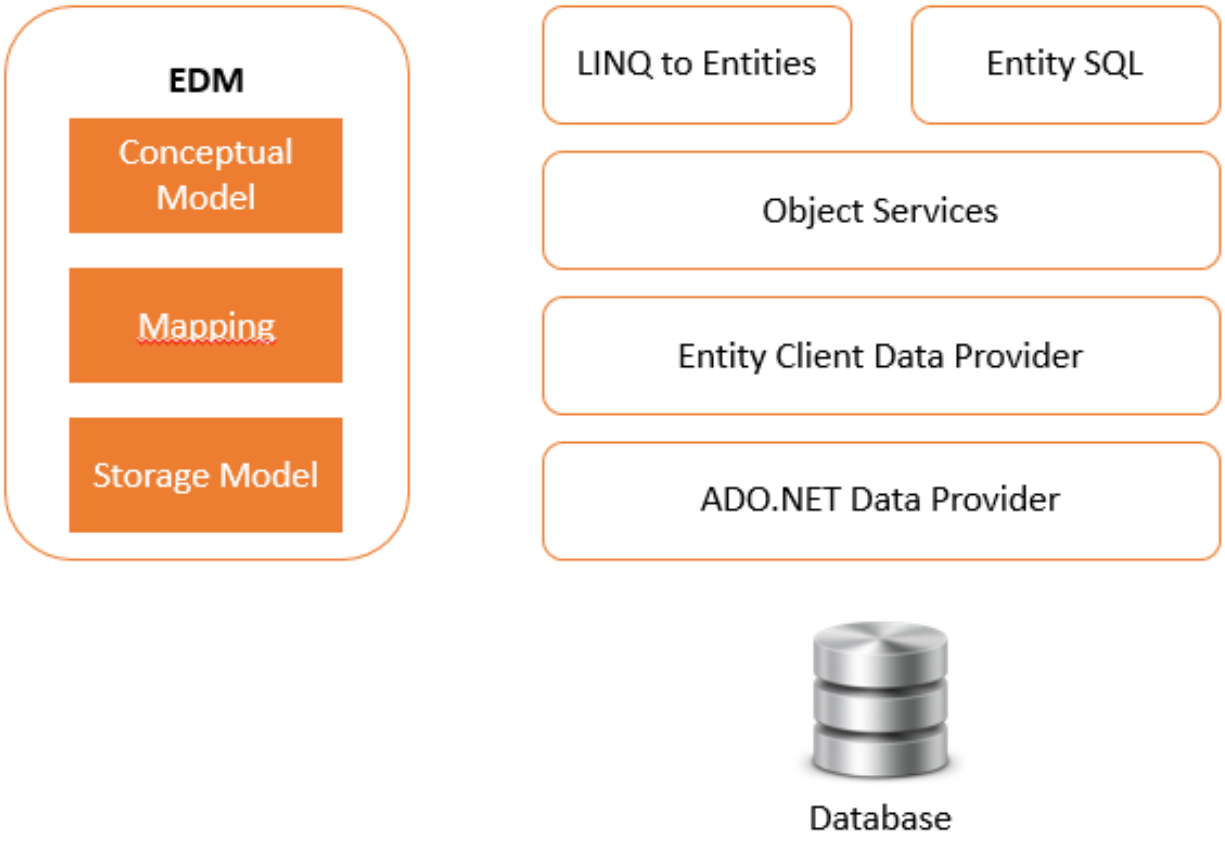


***Figura 10**: Architettura Entity Framework*

I componenti che costituiscono il framework sono:

- EDM (Entity Data Model): l'EDM è l'insieme del Conceptual Model, dello Storage Model e del Mapping.
- Conceptual Model: l'EF costruisce il modello concettuale a partire dalle classi di dominio e le loro relazioni. Il tutto è indipendente da come vengono progettate le tabelle del database.
- Storage Model: rappresenta la progettazione del database, includendo tabelle, view, stored procedure, relazioni e chiavi.
- Mapping: indica come mappare il Conceptual Model nello Storage Model.
- LINQ to Entities: è un linguaggio per scrivere query nei confronti del modello degli oggetti. Queste query ritornano entità definite nel conceptual model.
- Entity SQL: è un nuovo linguaggio per scrivere query introdotto con l'EF 6.
- Object Service: serve a tradurre i dati ritornati dal livello sottostante, l'entity client data provider, in oggetti istanza delle entità.
- Entity Client Data Provider: ha il compito di convertire le query LINQ-to-Entities o Entity SQL in query SQL comprensibili dal database sottostante. Comunica con il livello sottostante, l'ADO.NET data provider, dal quale riceve in cambio i dati provenienti dal database.
- ADO.NET Data Provider: è il livello che accede al database utilizzando l'ADO.NET standard.

Il primo compito principale dell'Entity Framework è costruire l'Entity Data Model, tramite il quale è possibile eseguire le operazioni CRUD, ottenendo query SQL a partire dalle query LINQ e trasformando i risultati provenienti dal database in oggetti entità. Le modifiche al database richieste dalle query verranno rese persistenti chiamando il metodo *SaveChanges()*.

4.5.4 Classe context

Quando si decide di utilizzare l'Entity Framework bisogna definire una classe context che estende la classe DbContext fornita dal framework. Questa classe è fondamentale perché permette di gestire una sessione con il database sottostante. Un esempio di classe context è il seguente:

```
public class SchoolContext : DbContext
{
    public SchoolContext()
    {

    }

    public DbSet<Studente> Studenti { get; set; }
    public DbSet<IndirizzoStudente> IndirizziStudenti { get; set; }
    public DbSet<Votazione> Votazioni { get; set; }
}
```

In questo esempio, la classe context definisce un set di entità per ciascuna delle entità Studente, IndirizzoStudente e Votazione.

Tramite questa classe è possibile richiedere o salvare dati nel database, configurare le classi di dominio, configurare il mapping e molto altro.

4.5.5 Entità

L'Entity Framework considera come entità ogni classe nel dominio dell'applicazione che viene aggiunta come una proprietà di tipo *DbSet<TEntity>* all'interno della classe context. In questo modo il framework può effettuare il mapping tra ogni entità e una tabella nel database e tra ogni proprietà di un'entità e una colonna nel database. Ricontrollando l'esempio nel paragrafo precedente, si intuisce come le classi Studente, IndirizzoStudente e Votazione siano delle classi di dominio che diventano entità nel momento in cui si includono nella classe context le proprietà *DbSet<Studente>, DbSet<IndirizzoStudente>* e *DbSet<Votazione>*.

Un'entità può avere due tipi di proprietà:

- Proprietà scalare: le proprietà costituite da tipi primitivi sono chiamate proprietà scalari. Una proprietà scalare viene mappata in una singola colonna all'interno del database.
- Proprietà di navigazione: le proprietà di navigazione indicano le relazioni con altre entità. È possibile suddividere le proprietà di navigazione in Reference e Collection:
    - Proprietà di navigazione reference: queste proprietà permettono di indicare una molteplicità di uno (1). Quindi una proprietà di tipo entità all'interno di un'entità è una proprietà di navigazione reference.
    - Proprietà di navigazione collection: queste proprietà permettono di indicare una molteplicità di molti (*). Una proprietà di tipo collection all'interno di un'entità è una proprietà di navigazione collection.

4.5.5.1 Tipi di entità

L'Entity Framework prevede due tipi di entità: le entità POCO e le entità Dynamic Proxy.

Un'entità POCO (Plain Old CLR Object) è come ogni altra classe .NET., quindi non dipende da nessuna classe base specifica del framework. Le entità POCO non hanno alcuna informazione riguardo la persistenza. Un esempio di entità POCO è il seguente:

```
public class Studente
{
    public int IDStudente { get; set; }
    public string Nome { get; set; }
    public DatetTime? DataDiNascita { get; set; }

    public IndirizzoStudente IndirizzoStudente { get; set; }
    public Votazione Votazione { get; set; }
}
```

Le entità Dynamic Proxy sono classi proxy generate a runtime che fungono da contenitore per le entità POCO. Affinché un'entità POCO possa essere eletta a dynamic proxy deve soddisfare i requisiti seguenti:

- La classe deve essere dichiarata con accesso public.
- La classe non deve essere sealed.

- La classe non deve essere astratta.
- Ogni proprietà di navigazione deve essere public e virtual.
- Ogni proprietà di navigazione collection deve essere di tipo *ICollection<T>*.
- L'opzione *ProxyCreationEnabled* all'interno della context class deve essere settata a true (di default accade proprio questo).

Un esempio di entità POCO che rispetta tutti questi requisiti e che quindi a runtime può diventare un'entità dynamic proxy è il seguente:

```
public class Student
{
    public int IDStudente { get; set; }
    public string Nome { get; set; }
    public DatetTime? DataDiNascita { get; set; }

    public virtual IndirizzoStudente IndirizzoStudente { get; set; }
    public virtual Votazione Votazione { get; set; }
}
```

Il dynamic proxy, abilitato di default per ogni entità, può essere disabilitato settando *context.Configuration.ProxyCreationEnabled = false* all'interno della context class.

Questo tipo di entità viene utilizzato quando si vuole sfruttare il lazy loading, Lazy loading vuol dire ritardare il caricamento delle proprietà di navigazione di un'entità fino a quando non si fa specificatamente accesso ad essa. Considerando l'esempio soprastante, al caricamento di un'entità Studente da parte della classe context non corrisponde automaticamente il caricamento della sua proprietà di navigazione IndirizzoStudente: questa proprietà verrà caricata solo quando viene fatto accesso ad essa.

L'altro approccio possibile è quello dell'eager loading, in cui durante il caricamento di un'entità viene specificato anche l'elenco delle proprietà di navigazione da caricare utilizzando un'unica query. Questo vuol dire che il framework non avrà bisogno di eseguire una query separata quando verrà richiesto l'accesso ad una delle proprietà di navigazione già caricate tramite eager loading. L'eager laoding è ottenuto utilizzando il metodo *Include()*. Un esempio è il seguente:

```
var stud1 = ctx.Students
                    .Include("Standard")
                    .Where(s => s.StudentName == "Bill")
                    .FirstOrDefault<Student>();
```

4.5.5.2 Stato delle entità

L'entity framework tiene traccia dello stato di ciascuna entità durante il suo ciclo di vita, cioè tiene traccia delle proprietà modificate per ogni entità. Questa funzionalità prende il nome di Change Tracking. In base all'operazione a cui è stata sottoposta un'entità, il suo stato potrebbe essere quello di *aggiunta, modificata, eliminata, non modificata* e *detached*.

Il cambiamento di stato da *non modificata* a *modificata* è l'unico ad essere gestito automaticamente dalla classe context. Gli altri cambiamenti vengono effettuati esplicitamente con uno dei metodi della classe DbContext o DbSet.

Nel momento in cui viene chiamato il metodo *context.SaveChanges()* vengono eseguiti i comandi INSERT, UPDATE o DELETE sulla base dello stato di un'entità. In particolare per le entità che si trovano nello stato *aggiunta* verrà eseguito il comando INSERT, per quelle che si trovano nello stato *modificata* viene eseguito il comando UPDATE e per quelle che si trovano nello stato *eliminata* viene eseguito il comando DELETE. Non è tenuta alcuna traccia, invece, per le entità che si trovano nello stato *detached*.

4.5.6 Approcci per lo sviluppo

Durante lo sviluppo di un'applicazione che fa utilizzo dell'Entity Framework è possibile utilizzare uno tra gli approcci Database-First, Code-First e Model-First.

Nell'approccio Database-First si parte da un database esistente per generare la classe context e le entità.

L'approccio Code-First viene utilizzato quando non si ha a disposizione un database per l'applicazione. In questo caso si comincia scrivendo le classi di dominio (entità) e la classe context e il database verrà automaticamente generato a partire da esse. L'approccio utilizzato nello sviluppo dell'applicazione oggetto di questo elaborato di tesi è proprio questo. Infatti, come verrà spiegato nel corso del capitolo successivo, l'applicazione è stata sviluppata seguendo i principi del Domain-Driven Design (DDD). Quando si seguono questi principi di progettazione viene naturale utilizzare l'approccio Code-First.

Nell'approccio Model-First si comincia descrivendo le entità, le relazioni e le gerarchie di eredità direttamente con il visual designer presente in Visual Studio. Dal modello visuale così ottenuto vengono generate le entità, la classe context e il database.

5. Implementazione dell'applicazione

In questo capitolo verrà descritta l'implementazione dell'applicazione oggetto di questa tesi, descrivendo gli approcci seguiti durante lo sviluppo, come il Domain-driven design, e mostrando porzioni di codice relativo alle varie funzionalità implementate.

5.1 Single-page application

Per offrire all'utente la migliore user-experience possibile si è deciso di implementare una single-page application (SPA). Le single-page application appaiano molto simili alle applicazioni desktop perché non interrompono l'user-experience durante la navigazione tra una pagina e l'altra, ma riscrivono la pagina corrente senza richiederne una nuova al server.

In una single-page application tutto il codice necessario, come per esempio quello HTML, JavaScript e CSS, viene scaricato per intero durante il primo caricamento della pagina. In seguito all'interazione con l'utente è possibile caricare in modo dinamico le risorse richieste e aggiungerle alla pagina. Solitamente l'accesso a queste risorse avviene sfruttando AJAX per connettersi alle API messe a disposizione da un servizio web. Questo vuol dire che nelle SPA parte della logica viene spostata dal server al client.

Una SPA non ha mai bisogno di ricaricarsi ed è efficiente dal punto di vista della larghezza di banda utilizzata, visto che dopo il caricamento iniziale dell'applicazione vengono trasferititi solo dati in formato JSON o XML e non intere pagine HTML.

La realizzazione di una single-page application è possibile utilizzando un framework come Angular, il cui funzionamento è stato descritto nel capitolo precedente.

5.2 Struttura dell'applicazione Angular per il front-end

Per quanto riguarda la progettazione della logica client-side, l'applicazione è stata suddivisa in diversi moduli, uno per ogni funzionalità offerta, sfruttando al massimo le potenzialità fornite da Angular per la realizzazione di applicazioni modulari.

La "Figura 11" mostra i moduli che sono contenuti all'interno del modulo principale AppModule. Considerando i requisiti che sono stati seguiti per lo sviluppo dell'applicazione, le funzionalità offerte possono essere suddivise in funzionalità relative alla dashboard, alle anagrafiche, alle associative, alle tabelle di dominio e alle consuntivazioni. Per ognuna di queste funzionalità è stato realizzato un modulo, così da organizzare nel modo migliore possibile il codice scritto. Oltre ai moduli associati a queste funzionalità ne è stato aggiunto un altro, SharedModule che serve a rendere disponibile a tutti gli altri moduli un sistema di notifiche per la segnalazione di eventuali messaggi di errore ricevuti dal server che si possono presentare durante l'utilizzo dell'applicazione.

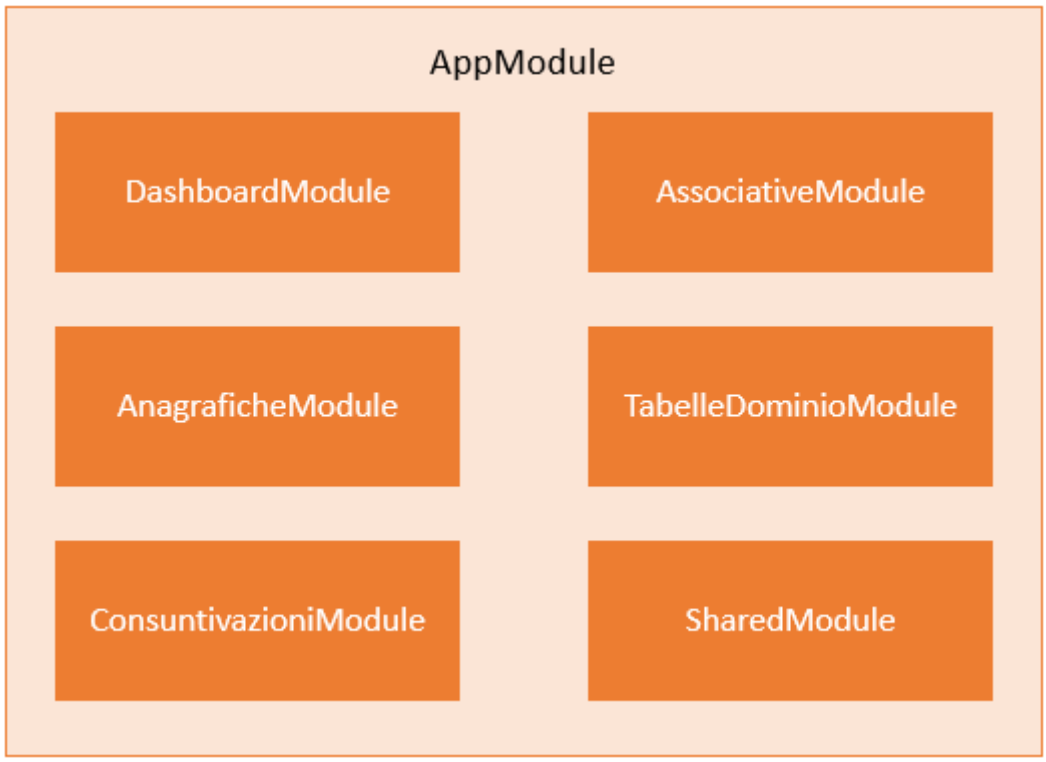


Figura 11: Moduli dell'applicazione Angular

Il modulo principale ingloba anche un componente PageNotFound che viene chiamato in causa ogni qual volta l'utente digiti un URL che non può essere gestito da alcun componente. Infatti, il routing attraverso questi moduli è gestito dall' AppRoutingModule, che ha il compito di visualizzare il componente PageNotFound quando necessario.

Un insieme di service permette al modulo principale di gestire la configurazione dell'applicazione, l'autenticazione dell'utente e le autorizzazioni associate ad un utente autenticato.

5.3 Struttura della logica server-side

Per quanto riguarda la progettazione della logica del web service, che espone le API contattate dall'applicazione Angular, è stato seguito un approccio Domain-driven design. Prima di procedere oltre è utile comprendere in cosa consiste questa tecnica di progettazione.

5.3.1 Domain-driven design

Il Domain-driven design può essere considerato come un approccio allo sviluppo di software scalabile che risolve le complessità causate da un dominio applicativo di dimensioni elevate e continuamente in evoluzione.

Il Domain-driven porta a porre attenzione principalmente al dominio e alla logica di dominio. La progettazione deve quindi essere basata su un modello del dominio. È fondamentale inoltre

un'attiva cooperazione tra gli sviluppatori e gli esperti di dominio in modo da costituire un modello che si evolve sulla base dei problemi che possono emergere durante la progettazione.

5.3.1.1 Concetti fondamentali

I concetti fondamentali relativi al Domain-driven design sono il dominio, il modello, l'ubiquitous language e il contesto.

Con il termine **dominio** si indica l'area delle conoscenze in cui agisce la logica dell'applicazione.

Il **modello** è un insieme di astrazioni che descrive aspetti determinati del dominio e può essere usato per risolvere problemi associati al dominio stesso. Quindi il modello serve ad individuare lo scopo dell'applicazione e la scelta di un modello piuttosto che un altro dipende proprio dallo scopo.

Il termine **contesto** indica l'ambiente in cui compare un termine e che ne determina il suo significato. Bisogna determinare esplicitamente il contesto all'interno del quale viene applicato il modello. Ed è fondamentale definire i confini di questo contesto in modo da rendere estranea l'applicazione da tutte le complessità circostanti che non fanno parte dello scopo per cui l'applicazione viene progettata.

L'**ubiquitous language** è la semantica definita attorno al modello del dominio e utilizzata da tutti i membri del team per identificare i problemi che devono essere risolti nel corso dell'implementazione. Quando ci si trova di fronte alla progettazione di un software, il punto di partenza è sempre lo stesso: gli sviluppatori conoscono il linguaggio di programmazione da utilizzare, i concetti fondamentali della programmazione a oggetti, il linguaggio UML (per modellare il dominio), ma non conoscono nulla del dominio sottostante all'applicazione. Il ponte tra gli sviluppatori e la conoscenza del dominio è costituito dagli esperti di dominio: è fondamentale quindi una comunicazione attiva tra sviluppatori e esperti di dominio in modo da definire un modello coerente per l'applicazione sviluppata. L'ubiquitous language fa sì che queste due tipologie di interlocutori possano parlare la stessa lingua. I termini definiti in questo modo devono essere utilizzati in modo coerente nella documentazione, nel codice e anche nelle discussioni tra tutti i membri del team. Se gli sviluppatori iniziano ad usare dei termini non conformi a quelli definiti con gli esperti di dominio, allora è probabile che gli sviluppatori non abbiano compreso appieno il dominio e quindi che l'implementazione si stia allontanando dal dominio originale.

È evidente che comprendere appieno il dominio e la sua logica non è un'operazione semplice e diretta, ma richiede tempo. Ogni qual volta si acquisisce una nuova informazione si deve valutare un eventuale attività di refactoring in modo da allineare il codice con le nuove informazioni acquisite. Solitamente questo processo di comprensione del dominio è guidato da eventuali errori nel modello corrente. Solo man mano che si avanza nello sviluppo la visione del dominio diventa via via più chiara.

Per poter affrontare nel migliore dei modi eventuali attività di refactoring il miglior alleato per gli sviluppatori è rappresentato dai test automatizzati.

5.3.1.2 Elementi fondamentali

Nell'ambito del Domain-driven design, è fondamentale realizzare un modello che colga l'essenza del dominio e che sia indipendente dalle tecnologie che potrebbero causare un'obsolescenza prematura. Il modello non deve presentare delle ridondanze inutili, deve essere costituito da componenti autodescrittivi e deve rispecchiare la comprensione attuale del dominio.

Anche i domini piuttosto complessi possono essere strutturati in una serie di componenti di media o bassa complessità, ciascuno con i propri ruoli e responsabilità. Ciò determina una certa modularità per l'applicazione sviluppata, che quindi diventa più facile da testare e da mantenere.

I blocchi costitutivi principali del Domain-driven design sono le Entità, i Value Object, gli Aggregate e i Repository.

Le **entità** sono caratterizzate da una storia e da un'identità. Infatti, dopo la loro creazione nel sistema, si trasformano nel corso del tempo modificando il loro stato. Da qui deriva il termine storia. Inoltre, hanno sempre associato un concetto di identità che permette di distinguere un'entità da un'altra anche in presenza di attributi con gli stessi valori. Da qui deriva il termine identità. Un esempio di entità potrebbe essere la tipica classe che modella un ordine: dopo la creazione di una sua istanza, l'ordine si evolve nel corso del tempo attraversando vari stati, quali per esempio aperto, evaso, fatturato o annullato. Nonostante il salvataggio in un database necessiti della presenza di una chiave, un'entità ha un concetto di identità indipendente da questa chiave, come per esempio l'attributo numero ordine associato ad un oggetto di tipo ordine. Per modificare il proprio stato interno, un'entità espone degli opportuni metodi che quando vengono chiamati garantiscono che lo stato dell'entità rimanga integro anche dopo la modifica.

I **Value Object** modellano quegli oggetti che non hanno bisogno di evolversi e di preservare la propria identità. Per esempio, se si acquista un giornale ad un'edicola, non ha importanza se il giornale ricevuto sia il primo o l'ultimo della pila: quindi il concetto di identità non ha alcuna utilità. Allo stesso modo, non essendo interessati allo stato dei Value Object, essi saranno degli oggetti immutabili. I valori degli attributi di un Value Object possono essere settati solo tramite un costruttore all'atto della creazione dell'oggetto. Dal punto di vista della persistenza, il fatto che i Value Object non abbiano il concetto di identità non vuol dire che non possano essere associati ad un record su una tabella a parte in cui vengono salvati tutti i Value Object dello stesso tipo. Ciò che permette di distinguere un Value Object da un'entità è il modo in cui viene gestito il ciclo di vita.

Gli **Aggregate** raggruppano un insieme di classi del dominio che per poter implementare una certa funzionalità necessitano di essere legate insieme da un'entità padre (chiamata anche Aggregate Root). Un tipico caso in cui ha senso definire un Aggregate si ha considerando una coppia Ordine-VoceOrdine, poiché un ordine non ha abbastanza informazioni dettagliate se preso singolarmente e contemporaneamente una VoceOrdine non ha senso di esistere da sola se non all'interno di un Ordine. Si può individuare un aggregato analizzando un confine sensato per propagare una cancellazione (sia essa fisica che logica). Nel caso della coppia Ordine-VoceOrdine questo processo è semplice, perché banalmente la cancellazione di un Ordine implica la cancellazione di tutte le VoceOrdine ad esso associato e viceversa la cancellazione

di una VoceOrdine può implicare delle modifiche per l'Ordine all'interno del quale era contenuta.

Il compito fondamentale di un aggregato è gestire la consistenza dell'insieme di oggetti al suo interno. Quanto detto porta a effettuare delle considerazioni importanti. Innanzitutto, un Aggregate Root possiede un'identità a livello globale, mentre le entità al suo interno hanno un'identità valida solo all'interno dell'Aggregate. Inoltre, gli oggetti esterni all'aggregato non possono avere dei riferimenti a entità interne all'Aggregate. O meglio, è possibile mantenere dei riferimenti ad un'entità interna ottenendoli tramite una query o chiedendoli direttamente alla radice dell'aggregato, ma non è possibile modificarla. Per trasformare lo stato di un Aggregate si deve passare sempre dalla sua radice. Si potrebbero presentare dei casi in cui le responsabilità di un Aggregate Root diventano troppo ampie e il controllo della consistenza rischia di generare un eccessivo accoppiamento tra più classi. Di fronte queste situazioni si può pensare di creare una nuova classe per l'aggregato che non si riflette in una tabella nel database, ma permette di ottenere una migliore separazione delle responsabilità. Questa classe ha il compito esclusivo di controllare la consistenza dell'aggregato nel suo insieme, mentre ogni entità all'interno dell'Aggregate deve esclusivamente garantire la correttezza del suo stato interno.

Gli aggregati permettono così di separare il modello di dominio in regioni indipendenti tra di loro e ad ognuna di esse saranno associati determinati casi d'uso.

I **Repository** espongono dei metodi che permettono di leggere e salvare gli oggetti di dominio nel database sottostante, facendo in modo che l'utilizzo di storage alternativi sia semplice. Nell'ambito del Domain-driven design, i repository fanno parte del livello del dominio e non del livello di persistenza dei dati. Infatti, le operazioni per l'accesso ai dati sono delle operazioni fondamentali all'interno dell'applicazione sviluppata. Il repository nella maggior parte dei casi dipende da framework sottostanti che implementano il mapping object-relational, come l'Entity Framework, che invece sono confinati all'interno del livello di infrastruttura. Poiché il modello del dominio è indipendente dal tipo di storage utilizzato, è più semplice testarlo realizzando dei Mock per il Repository. Quanto detto è un aspetto molto importante, perché fa in modo che l'invecchiamento del modello realizzato non sia legato a quello della tecnologia utilizzata.

5.3.2 Il Domain-driven design nel contesto dell'applicazione sviluppata

Dopo aver analizzato i requisiti dell'applicazione si è cercato di seguire quanto più possibile i principi esposti dal Domain-driven design.

5.3.2.1 Modello del dominio

Quindi nella fase iniziale dell'implementazione della logica di back-end la concentrazione è stata posta sul dominio e sulle regole di business che caratterizzano gli oggetti al suo interno. Il modello del dominio che si adatta meglio allo scopo per il quale è stata realizzata l'applicazione è quello che è già stato presentato nel terzo paragrafo del terzo capitolo e che riproponiamo di seguito per comodità, adattando dei piccoli accorgimenti in ottica Domain-driven design.

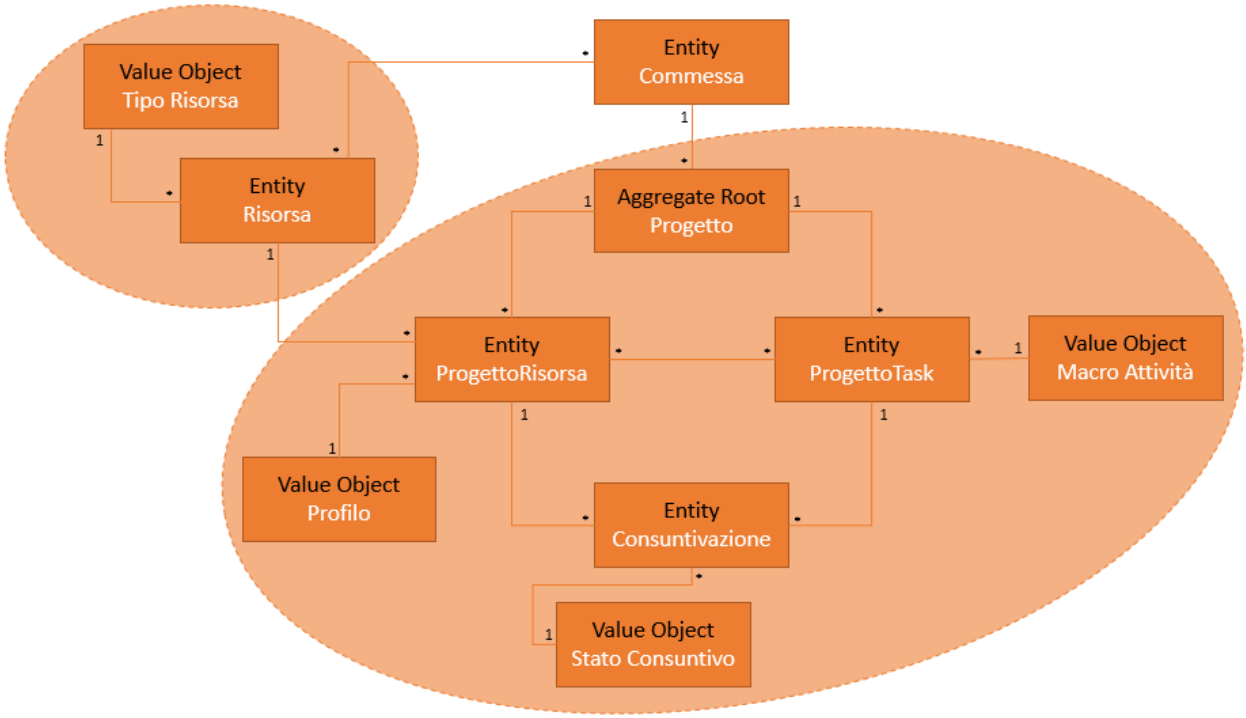


***Figura 12**: Modello del dominio*

È stato scelto come Aggregate Root l'entità Progetto e non Commessa perché la maggior parte delle operazioni avvengono a livello di Progetto, come l'associazione di una Risorsa o di un Task ad un Progetto. Infatti, un Progetto non avrebbe senso di esistere se non gli si potessero assegnare delle risorse e dei task, che a loro volta vengono svolti da una o più risorse appartenenti al progetto stesso. Allo stesso modo una Consuntivazione non ha senso di esistere da sola, ma deve essere collegata ad un task sul quale si è lavorato e ad una risorsa che ha svolto questo lavoro.

Tutte le operazioni che riguardano la modifica o l'aggiunta di un'entità facente parte dell'aggregato Progetto, come per esempio l'inserimento di una consuntivazione, passano attraverso la radice dell'aggregato che effettua i controlli necessari per determinare se l'operazione richiesta può essere effettuata senza alterare la consistenza dell'aggregato. Se per esempio si vuole aggiungere una nuova consuntivazione, bisogna verificare se la risorsa e il task ai quali la consuntivazione fa riferimento esistano davvero all'interno del progetto in cui si sta agendo. La "Figura 13" mostra il diagramma delle classi del dominio generato con l'utilizzo dell'IDE Visual Studio, con l'elenco delle proprietà e dei metodi di ogni classe.

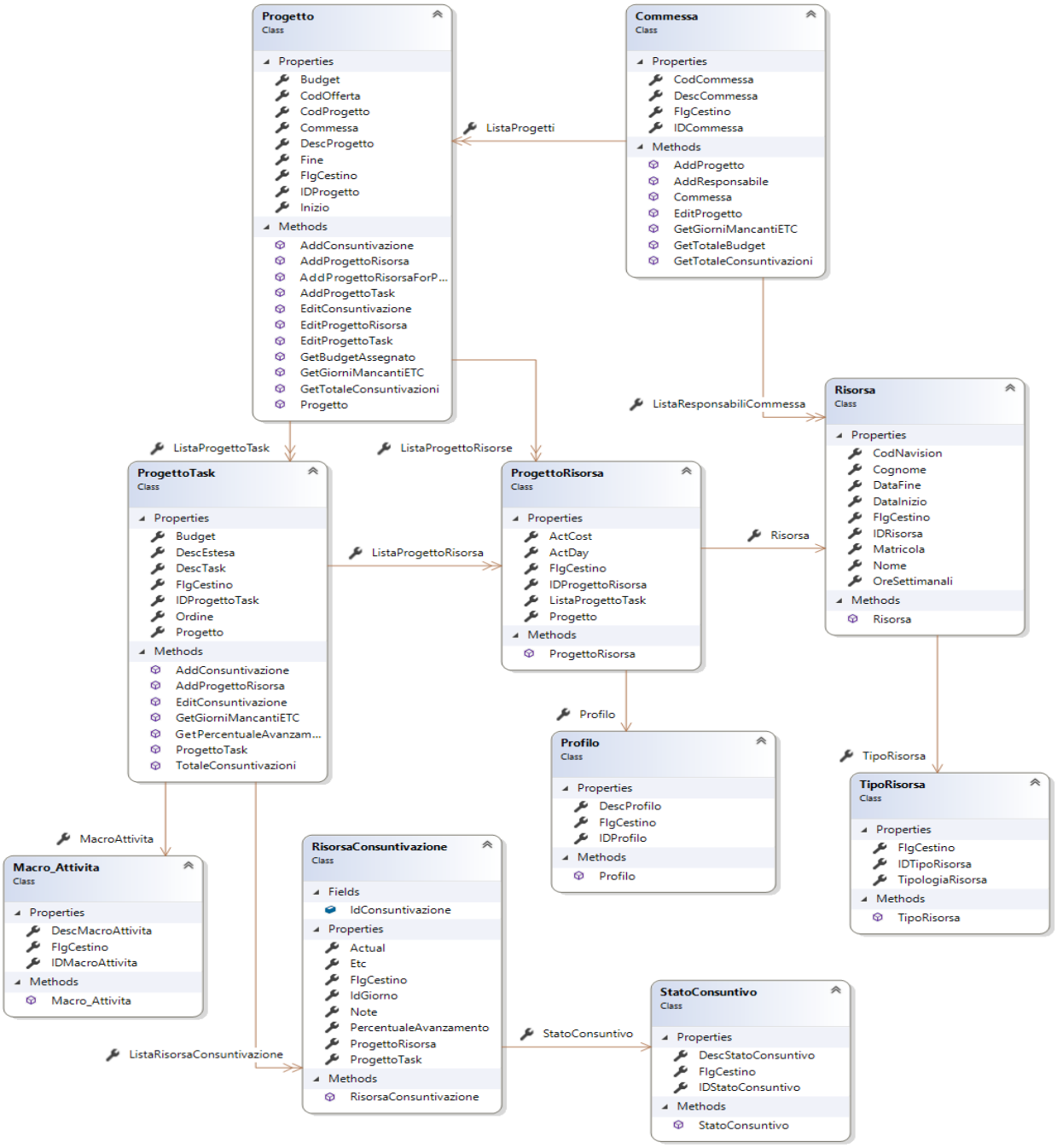


***Figura 13**: Diagramma delle classi*

La relazione tra l'entità Commessa e l'entità Risorsa è dovuta al fatto che ad una data commessa è possibile assegnare uno o più responsabili, i quali sono modellati dallo stesso oggetto Risorsa che viene utilizzato nell'associazione Progetto-Risorsa.

5.3.2.2 Principali regole di business

A questo punto verrà illustrato il comportamento dei metodi principali delle classi di dominio.

Il metodo per l'aggiunta di un task all'interno di un progetto è il seguente:

```
Progetto.cs
public ProgettoTask AddProgettoTask(ProgettoTask progettoTask)
{
	//controllo che non ci sia già Task un task con lo stesso id associato al
progetto
	var esisteGia = ListaProgettoTask.Where(pt => pt.DescTask ==
			progettoTask.DescTask).Count();
	if (esisteGia > 0)
	{
		throw new ProgettoTaskGiaEsistenteException();
	}
	//controllo che il progetto presente nel nuovo progettoTask sia proprio quello
che	//sta chiamando il metodo addProgettoTask
	bool progettoCorretto = (progettoTask.Progetto == this);
	if (!progettoCorretto)
	{
		throw new Exception();
	}
	ListaProgettoTask.Add(progettoTask);
	return progettoTask;
}
```

Come si può notare, affinché l'inserimento vada a buon fine, è necessario che l'Aggregate Root (il Progetto corrente) controlli che non ci sia già un task con la stessa descrizione al suo interno e che il riferimento all'oggetto Progetto contenuto nel progettoTask, ricevuto come parametro in ingresso, corrisponda proprio al Progetto che sta chiamando il metodo *AddProgettoTask*. Questi stessi controlli devono essere effettuati anche quando viene richiesta la modifica di un ProgettoTask.

Il metodo per l'aggiunta di un'associazione ProgettoRisorsa all'interno di un progetto, *AddProgettoRisorsa,* fa esattamente la stessa cosa, ricevendo però come argomento un oggetto di tipo ProgettoRisorsa e non ProgettoTask, a differenza di quanto visto nel metodo soprastante.

Per poter permettere ad una risorsa di operare su un task di un progetto viene utilizzato il metodo seguente:

```
Progetto.cs
public void AddProgettoRisorsaForProgettoTask(ProgettoTask progettoTask, short
	IDProgRisorsa)
{
	ProgettoRisorsa progRisorsa = ListaProgettoRisorse.FirstOrDefault(pr =>
			pr.IDProgettoRisorsa == IDProgRisorsa);
	if (progRisorsa == null)
	{
		throw new ProgettoRisorsaNotFoundException();
	}
	progettoTask.AddProgettoRisorsa(progRisorsa);
	progRisorsa.ListaProgettoTask.Add(progettoTask);
}
```

Dopo aver controllato che l'ID del ProgettoRisorsa ricevuto come argomento corrisponda veramente ad un ProgettoRisorsa presente nel Progetto, il compito dell'aggiunta del ProgettoRisorsa all'interno della lista di oggetti di tipo ProgettoRisorsa del ProgettoTask, viene delegato proprio al ProgettoTask ricevuto come primo argomento:

```
ProgettoTask.cs
public ProgettoRisorsa AddProgettoRisorsa(ProgettoRisorsa pr)
{
	//controllo che non ci sia già una risorsa con lo stesso id associata al task
```

```
        var progGiaInserito = ListaProgettoRisorsa.Where(p => (p.Risorsa.IDRisorsa ==
                        pr.Risorsa.IDRisorsa)).Count();
        if(progGiaInserito > 0)
        {
                throw new ProgettoRisorsaTaskGiaPresenteException();
        }
        ListaProgettoRisorsa.Add(pr);
        return pr;
}
```

Il ProgettoTask chiamato in causa deve controllare che al suo interno non esista già il ProgettoRisorsa ricevuto come parametro in ingresso.

Il metodo per l'aggiunta di una consuntivazione deve verificare che questa nuova consuntivazione si riferisca ad un ProgettoTask e ad un ProgettoRisorsa già associati al progetto:

```
Progetto.cs
public RisorsaConsuntivazione AddConsuntivazione(short IDCons, DateTime IDGiorno,
short  IDRisorsa, short IDTask, StatoConsuntivo Stato, decimal Actual, decimal Etc,
        decimal PercentualeAvanzamento, string Note, bool FlgCestino)
{
        var progettoTask = ListaProgettoTask.FirstOrDefault(x => x.IDProgettoTask ==
                        IDTask);
        if (progettoTask == null)
        {
                throw new ProgettoTaskNotFoundException();
        }
        var progettoRisorsa = ListaProgettoRisorse.FirstOrDefault(x =>
x.Risorsa.IDRisorsa         == IDRisorsa);
        if (progettoRisorsa == null)
        {
                throw new ProgettoRisorsaNotFoundException();
        }
        RisorsaConsuntivazione rc = new RisorsaConsuntivazione(
                        IDCons,
                        Stato,
                        IDGiorno,
                        progettoTask,
                        progettoRisorsa,
                        Actual,
                        Etc,
                        PercentualeAvanzamento,
                        Note,
                        FlgCestino);
        RisorsaConsuntivazione consInserita;
        consInserita = progettoTask.AddConsuntivazione(rc);

        return consInserita;
}
```

Se le due condizioni enunciate precedentemente sono rispettate allora si può chiamare in causa il ProgettoTask associato alla consuntivazione da inserire. Tale ProgettoTask inserirà la consuntivazione nella lista delle sue cosuntivazioni:

```
ProgettoTask.cs
public RisorsaConsuntivazione AddConsuntivazione(RisorsaConsuntivazione rc)
{
        //Non deve esistere un'altra risorsaConsuntivazione con lo stesso IdGiorno e
        //associata alla stessa risorsa in ListaRisorsaConsuntivazione
```

```
        var risGiaInserita = ListaRisorsaConsuntivazione.Where(r => (r.IdGiorno.Year ==
                        rc.IdGiorno.Year && r.IdGiorno.Month == rc.IdGiorno.Month &&
r.IdGiorno.Day              == rc.IdGiorno.Day && r.ProgettoRisorsa.Risorsa.IDRisorsa
==                                  rc.ProgettoRisorsa.Risorsa.IDRisorsa)).Count();
        if (risGiaInserita > 0)
        {
                throw new ConsuntivazioneGiaEsistenteException();
        }
        bool progTaskCorretto = (rc.ProgettoTask == this);
        if (!progTaskCorretto)
        {
                throw new Exception();
        }
        ListaRisorsaConsuntivazione.Add(rc);
        return rc;
}
```

Il ProgettoTask in esame deve controllare che tra le sue consuntivazioni non ce ne sia già un'altra associata allo stesso giorno e alla stessa Risorsa. L'univocità di una consuntivazione è infatti data dalla combinazione del giorno con il progetto, il task e la risorsa a cui fa riferimento la consuntivazione. Queste condizioni devono essere verificate anche quando si vuole modificare una consuntivazione già esistente.

Le entità Commessa, Progetto e ProgettoTask espongono anche dei metodi che permettono di conoscere l'andamento delle varie attività sia a livello di commessa, sia a livello di progetto e sia a livello di task. A livello di task i seguenti metodi calcolano rispettivamente il totale delle ore consuntivate e l'estimated time to complete (in giorni) del task stesso, sfruttando la lista delle consuntivazioni a esso associate:

ProgettoTask.cs
```
public decimal TotaleConsuntivazioni()
{
        //somma totale ore svolte nel task
        if (ListaRisorsaConsuntivazione.Count > 0)
        {
                return ListaRisorsaConsuntivazione.Sum(rc => rc.Actual);
        }
        else
        {
                return 0;
        }
}
public decimal GetGiorniMancantiETC()
{
        if (ListaRisorsaConsuntivazione.Count > 0)
        {
                return ListaRisorsaConsuntivazione.OrderByDescending(rc =>
                                rc.IdGiorno).ThenBy(rc => rc.Etc).ToList()[0].Etc/8;
        }
        else
        {
                return Budget;
        }
}
```

Il metodo per ottenere l'ETC del task ordina le consuntivazioni prima in ordine decrescente secondo il giorno e quindi in ordine crescente per il valore di ETC. Infatti, per calcolare la quantità di tempo necessaria a completare il task basta accedere a quelle consuntivazioni inserite per ultime in ordine di tempo. Poiché ci potrebbero essere più consuntivazioni per lo

stesso task nello stesso giorno (caso in cui più risorse hanno i permessi di lavorare sul task) si assume come valore di ETC quello più basso.

Queste stesse informazioni possono essere lette anche a livello di Progetto e di Commessa con i seguenti metodi:

Progetto.cs
```
public decimal GetTotaleConsuntivazioni()
{
	return ListaProgettoTask.Sum(tot => tot.TotaleConsuntivazioni());
} //somma totale ore svolte nel progetto

public decimal GetGiorniMancantiETC()
{
	return ListaProgettoTask.Sum(pt => pt.GetGiorniMancantiETC());
}
```

Commessa.cs
```
public decimal GetTotaleConsuntivazioni()
{
	return ListaProgetti.Sum(p => p.GetTotaleConsuntivazioni());
}
public decimal GetGiorniMancantiETC()
{
	return ListaProgetti.Sum(p => p.GetGiorniMancantiETC());
}
```

5.3.2.3 Testing del domain model

Le operazioni che possono essere svolte dagli oggetti del dominio sono state testate tramite l'utilizzo di unit test. Infatti, l'approccio Domain-driven design rende l'applicazione sviluppata più mantenibile e anche facilmente testabile. I metodi di test utilizzati seguono il pattern Arrange-Act-Assert. Con il termine Arrange si indica la preparazione delle condizioni per svolgere il test, con il termine Act si indica lo svolgimento del test e con il termine Assert si indica il confronto dei risultati ritornati dal test con i risultati attesi. Come esempio viene mostrato il metodo di test per l'inserimento di un nuovo task all'interno di un Progetto:

```
[TestMethod]
public void AddProgettoTaskInProgetto()
{
	//arrange
	DateTime data20_10_2017;
	DateTime data30_10_2018;
	Commessa Comm1 = new Commessa(1, "CMMSS1", "Commessa 1", false);
	Progetto Prog1 = new Progetto(1, "PRGTT1", "Progetto 1", false, Comm1,
	"Offerta1", data20_10, data30_10, 100);
	Progetto Prog2 = new Progetto(2, "PRGTT2", "Progetto 2", false, Comm1,
	"Offerta1", data20_10, data30_10, 110);
	Macro_Attivita MacroAttivita = new Macro_Attivita(1, "Analisi", false);
	ProgettoTask ProgTask1 = new ProgettoTask(1, Prog1, MacroAttivita, "task 1",
	false, 50, 1, null);
	ProgettoTask ProgTask2 = new ProgettoTask(2, Prog1, MacroAttivita, "task 2",
	false, 40, 2, null);
	//act
	var progettoTaskInserito1 = Prog1.AddProgettoTask(ProgTask1);
	var progettoTaskInserito2 = Prog1.AddProgettoTask(ProgTask2);
	//assert
```

```
        Assert.AreEqual(progettoTaskInserito1, ProgTask1, "L'inserimento di un
        progettoTask nuovo non deve generare errori");
        Assert.AreEqual(progettoTaskInserito2, ProgTask2, "L'inserimento di un secondo
        progettoTask nuovo non deve generare errori");
        Assert.AreEqual(Prog1.ListaProgettoTask[0], ProgTask1);
}
```

5.3.3 Architettura generale del servizio web

Dopo aver descritto l'implementazione del dominio si può passare ad analizzare l'architettura generale del servizio web e i livelli che la costituiscono, mostrando tutto ciò che ruota attorno al dominio stesso.

Il livello più alto è quello dei controller, che vengono chiamati in causa quando una nuova richiesta viene ricevuta dal client. Come mostrato nel capitolo precedente, in cui è stato descritto il funzionamento delle Web API 2, l'URL associato alla richiesta HTTP sopraggiunta permette di individuare il controller responsabile alla gestione della richiesta, mentre il metodo HTTP e gli eventuali parametri di query permettono di invocare uno specifico action method del controller.

Il livello successivo è quello dei service. Infatti, tramite depency injection, viene iniettata in ogni controller l'interfaccia di un opportuno service, dotato di una serie di metodi che possono essere invocati per elaborare la richiesta ricevuta. Ad ogni classe del dominio corrisponde un service.

Per avere una maggiore *separation of concerns* è stato seguito il pattern Command Query Responsability Segregation (CQRS). Tale pattern impone una netta distinzione tra il modello concettuale utilizzato per la visualizzazione dei dati e quello utilizzato per l'aggiornamento degli stessi, indicati rispettivamente con i termini Query e Command. Nell'applicazione oggetto di questa tesi le operazioni di lettura dei dati vengono gestite da componenti che implementano l'interfaccia *IQueryHandler*, mentre le operazioni di modifica/inserimento dei dati sono gestite dai Repository. I Query Handler non vengono invocati direttamente, ma tramite un componente chiamato Query Dispatcher. Questo componente riceve un oggetto che implementa l'interfaccia *IQuery* (contenente i parametri della richiesta) e, in base al tipo di dato (implementazione dell'interfaccia *IQueryResult*) che si aspetta di ricevere il service, invoca il Query Handler incaricato di gestire questa specifica coppia. Per molte classi del dominio esistono più terne Query - Query Result - Query Handler, perché le stesse informazioni potrebbero essere presentate in modo diverso al client a seconda della pagina in cui si trova l'utente dell'applicazione. Per quanto riguarda i Repository, ne è stato implementato uno per ognuna delle funzionalità associata alle tabelle di dominio (Profilo, Macro-Attività, TipoRisorsa e StatoConsuntivo), uno per l'Aggregate Root Progetto, uno per l'entità Commessa e infine uno per l'entità Risorsa.

Ad ogni service vengono iniettate sia l'interfaccia del Repository adatto a gestire il tipo di dato in esame, sia l'interfaccia del Query Dispatcher. Quindi, uno dei compiti dei service è quello di determinare se una data richiesta debba essere gestita dal Query Handler o dal Repository che gli sono stati iniettati. Nel primo caso il Query Handler accedere direttamente al database per leggere le informazioni richieste senza popolare tutti gli oggetti del dominio, operazione che avrebbe un impatto pesante in termini di prestazioni. Nel secondo caso invece, il Repository potrebbe popolare gli oggetti del dominio nel caso in cui ci siano regole di business

da applicare prima di poter rendere le modifiche persistenti sul database. La “Figura 14” sintetizza quanto detto.

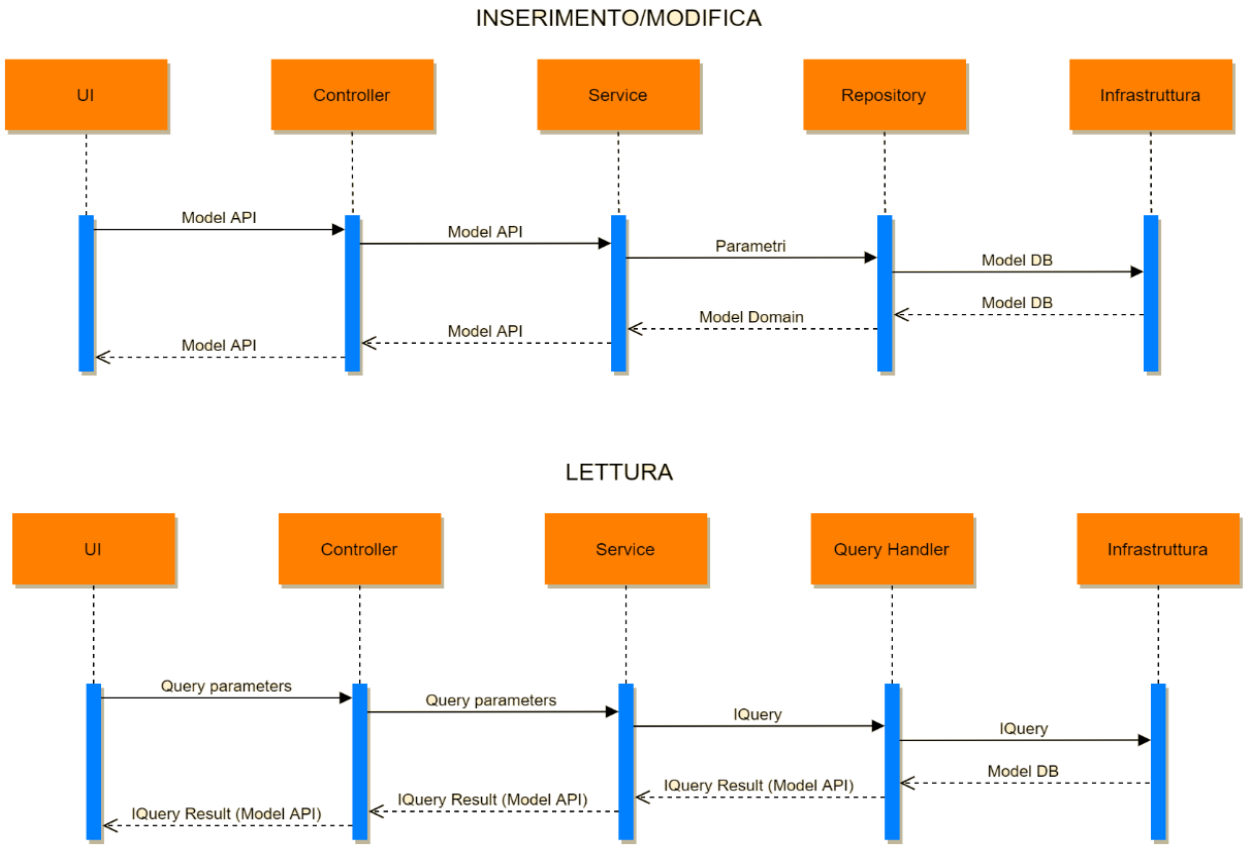


***Figura 14**: Architettura applicazione*

Nel caso di aggiunta o modifica dei dati il controller riceve un oggetto, che per convenzione viene chiamato Model API, e lo passa al service. Il service estrae gli attributi di questo oggetto e li passa al repository. Quando è richiesta un’operazione di modifica dei dati che riguardano l’AggregateRoot o una delle entità al di sotto di esso, si chiama sempre lo stesso Repository: ad un AggregateRoot corrisponde un solo Repository responsabile di gestire le modifiche a tutte le entità appartenenti a questo aggregato. Questo Repository costruisce e popola gli oggetti di dominio a partire dall’AggregateRoot. Quindi, con i parametri ricevuti dal service soprastante crea un oggetto, il cui tipo è la classe di dominio corrispondente all’entità che si vuole modificare, e prova a effettuare l’operazione richiesta invocando il metodo dell’AggregateRoot associato a questa operazione. Se non viene sollevata alcuna eccezione allora non è stata violata alcuna regola di business dell’aggregato. Il repository crea un oggetto di database corrispondente all’entità su cui si sta agendo, che per convenzione viene chiamato Model DB, rende il dato persistente nel database sfruttando l’Entity Framework e ritorna al service un oggetto, il cui modello è quello del dominio, corrispondente all’oggetto Model DB appena inserito nel database. Il service crea un oggetto Model API corrispondente all’oggetto di dominio che gli è stato ritornato dal repository e lo passa al controller.

È importante precisare che le classi di dominio non conoscono nulla né riguardo i repository, né riguardo i service né riguardo l'Entity Framework.

Nel caso di lettura il controller estrae i parametri dall'URL, se presenti, e li passa al service che crea un oggetto implementazione dell'interfaccia *IQuery*. A questo punto viene invocato il Query Handler che accede al database filtrando i dati a seconda dei parametri di query ricevuti e ritorna al service una lista di oggetti implementazione dell'interfaccia *IQueryResult*. Il service passa questa lista al controller che converte automaticamente i dati in JSON e li inoltra al client.

5.4 Gestione dell'autenticazione e delle autorizzazioni

Prima di descrivere l'implementazione delle varie funzionalità fornite dall'applicazione verrà analizzato il modo in cui vengono gestite l'autenticazione di un utente e le autorizzazioni a lui associate.

5.4.1 Autenticazione

Il fatto che l'applicazione sia destinata ad un uso esclusivamente all'interno dell'intranet dell'azienda in cui è stata svolta la tesi semplifica di molto la gestione dell'autenticazione, perché basta l'utilizzo dell'autenticazione Windows. Per fare ciò è necessario che il server in cui verrà ospitata l'applicazione sia abilitato al supporto dell'autenticazione Windows e che il file di configurazione dell'applicazione che espone le API contenga le seguenti linee di codice:

```
Web.config
<system.webServer>
  <security>
    <authentication>
      <anonymousAuthentication enabled="false" userName="" />
      <windowsAuthentication enabled="true">
      </windowsAuthentication>
    </authentication>
  </security>
</system.webServer>
```

Lato client, invece, l'header di ogni richiesta HTTP che viene inviata al web service deve avere la proprietà *withCredentials* settata a true. Per ottenere ciò senza duplicazione di codice, l'applicazione Angular è dotata di un HTTP interceptor che intercetta tutte le richieste in uscita e ne modifica l'header in modo da abilitare l'invio delle credenziali.

5.4.2 Autorizzazioni

Più complessa è invece la gestione dei permessi associati agli utenti dell'applicazione. Durante il caricamento dell'applicazione Angular, precisamente nella fase di inizializzazione del Component associato all'AppModule, una richiesta HTTP viene inviata al server ed in particolare al controller *WithAuthController* in modo da verificare se l'utente è abilitato ad utilizzare l'app. Il controller chiama in causa il service *AuthorizationService.* Questo crea un oggetto di tipo *IQuery* in cui l'unica proprietà è la matricola dell'utente (ricavata dall'autenticazione Windows) che sta provando ad utilizzare l'applicazione. Il Query

Dispatcher iniettato nel service invoca il Query Handler responsabile della gestione di questo tipo di richieste. Il Query Handler scelto controlla se le credenziali ricevute sono presenti all'interno della tabella degli utenti autorizzati all'uso dell'applicazione e in caso negativo invia una risposta di tipo 401 Not Authorized. Invece, in caso positivo recupera il ruolo dell'utente e l'insieme di permessi e voci di menu associati a questo ruolo in modo che il controller possa inviare tutte queste informazioni al client.

5.4.2.1 Voci di menu

Le voci di menu mostrate all'utente vengono costruite dinamicamente in base al ruolo sfruttando le potenzialità di Angular. Se l'utente corrente è associato al ruolo di Amministratore le voci di menu principali e secondarie che potrà vedere saranno come nella "Figura 15". Se invece ha il ruolo di Responsabile vedrà le stesse voci di menu mostrate nella "Figura 16". Infine, se ha il ruolo di Developer vedrà le voci di menu principali "Dashboard" e "Consuntivazione", ma nessuna voce di "Configurazione".

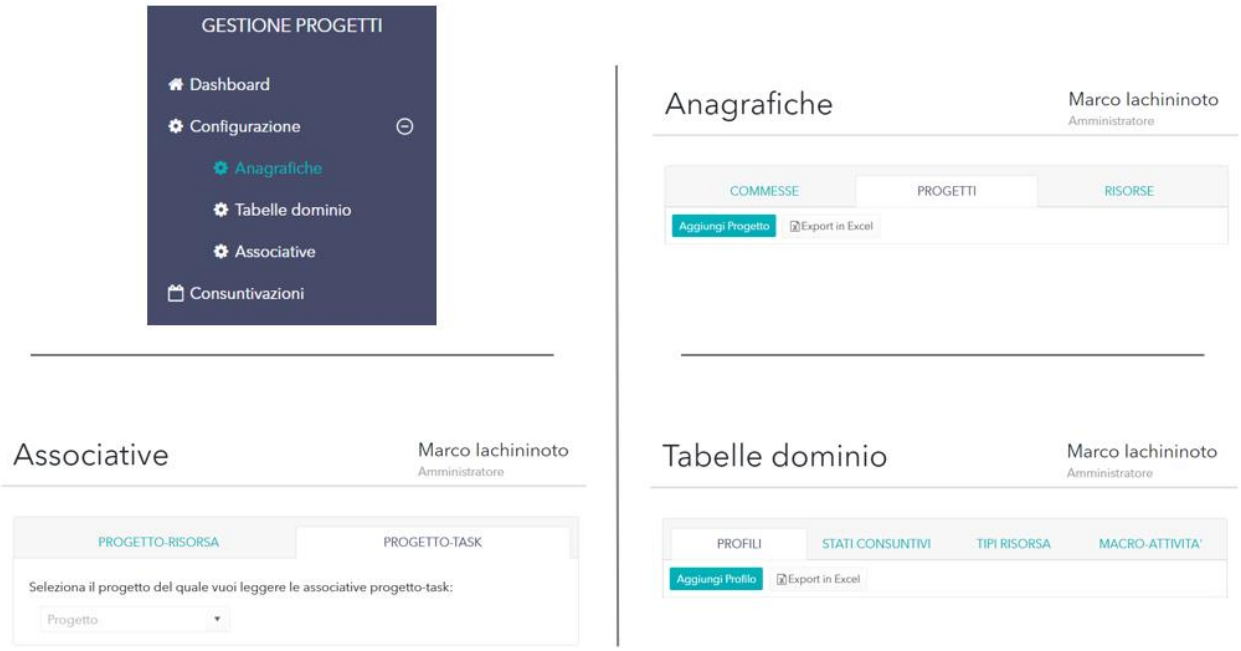


***Figura 15**: Voci di menu per utenza amministratore. In alto a sinistra quelle principali; in alto a destra le voci di menu secondarie associate al menu Anagrafiche; in basso a sinistra le voci di menu secondarie associate al menu Associative; in basso a destra le voci di menu secondarie associate al menu tabelle di dominio*

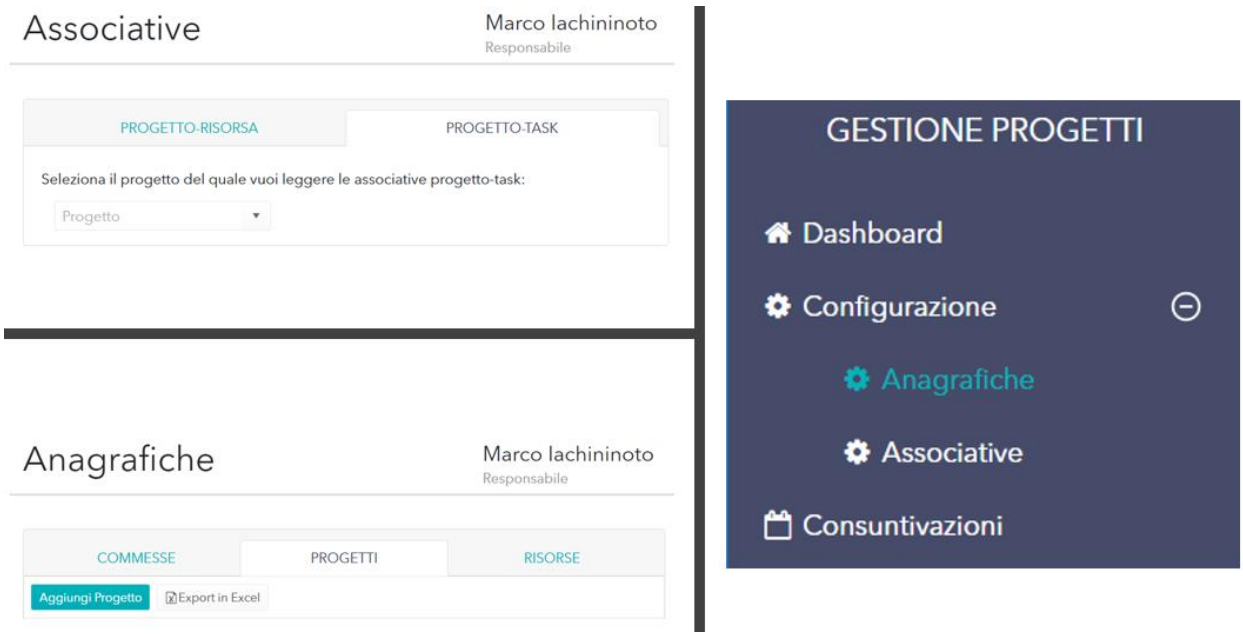


***Figura 16**: Voci di menu per utenza responsabile. A destra quelle principali; in alto a sinistra quelle secondarie associate al menu Associative; in basso a sinistra quelle secondarie associate al menu Anagrafiche*

5.4.2.2 Controllo permessi nel front-end

Per evitare che un utente autenticato acceda a una funzionalità per cui non possiede i permessi, cioè di cui non può vedere la voce di menu, inserendo manualmente l'URL della funzionalità nella barra degli indirizzi, si utilizza il meccanismo delle Route Guards offerto da Angular. Infatti, ogni accesso ad un componente corrispondente ad un certo URL viene temporaneamente bloccato per permettere al Route Guard di controllare se tra i permessi che sono stati ricevuti dal server durante il primo accesso all'applicazione c'è anche quello che permette l'accesso al componente desiderato. Qui di seguito viene mostrato il codice che effettua questi controlli per le voci di menu principali (in modo analogo avviene il controllo per le voci di menu secondarie):

```
auth-guard.service.ts
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> {
  var url: string = state.url.toLocaleLowerCase();
  switch(url){
    case "/dashboard": { //per l'accesso alla dashboard serve il permesso MN_HOME
      this.permission = "MN_HOME";
      this.errorMessage = "visualizzazione della dashboard";
      break;
    }
    case "/consuntivazioni": {
      this.permission = "MN_CONS_R";
      this.errorMessage = "visualizzazione della consuntivazione";
      break;
    }
```

```
    case "/anagrafiche": {
      this.permission = "SM_ANAG";
      this.errorMessage = "visualizzazione delle anagrafiche";
      break;
    }
    case "/associative": {
      this.permission = "SM_ASS";
      this.errorMessage = "visualizzazione delle associative";
      break;
    }
    default: return Observable.of(true);
  }
  return this.authSvc.checkPermission(this.permission, this.errorMessage);
  //authSvc è il service per l'autenticazione iniettato tramite dependency
  injection.
}
```

authentication.service.ts

```
checkPermission(permission: string, errorMessage: string): Observable<boolean>{
  //getUser() contatta il server per ottenere i permessi e le voci di menu
  //dell'utente solo al primo accesso, altrimenti ritorna le informazioni
  //delll'utente già salvate all'interno dell'oggetto authRslt.
  return this.getUser().map(() => this.check(permission, errorMessage));
}
private check(permission: string, errorMessage: string){
  const alert = this.injector.get(AlertService); //service che visualizza
notifiche
  if (this.authRslt.length === 0){ //nessun profilo associato all'utente
    alert.error("Non si è autorizzati ad utilizzare l'applicazione");
    return false;
  }
  var permissionIndex: number;
  permissionIndex = this.authRslt[0].PermissionList.findIndex(p =>
    p.CodPermission === permission); //Un utente ha sempre un solo profilo quindi
    //accedo al primo elemento dell'array authRslt
  if (permissionIndex === -1){ //Non si hanno i permessi per la funzionalità
    this.router.navigate(['/consuntivazioni']); //Redirect verso una funzionalità
    //accessibile da qualsiasi dei tre ruoli
    alert.error("Non si dispone dei permessi per la " + errorMessage); //mostra il
    //messaggio di errore all'utente
    return false;
  }
  return true;
}
```

Se la chiamata al metodo *CanActivate* ritorna true allora la navigazione può procedere, altrimenti la navigazione viene stoppata ed eventualmente l'utente viene reindirizzato verso un altro componente per cui dispone i permessi.

5.4.2.3 *Controllo permessi nel back-end*

Per evitare che un utente all'interno dell'intranet possa effettuare delle operazioni illecite utilizzando per esempio uno dei tanti strumenti disponibili per il testing di API, tutti gli action method di ogni controller sono protetti da un filtro che controlla i permessi dell'utente che ha richiesto una determinata operazione. Per esempio, l'operazione di aggiunta di una nuova commessa (operazione che può essere effettuata solo da un amministratore) è protetta in questo modo:

```
[CustomAuthorize("SN_ANAG_COMM_ADD")]
public IHttpActionResult Post([FromBody]CommessaAPI commessa) { … }
```

L'implementazione del filtro estende la classe AuthorizeAttribute ed effettua un override del metodo *OnAuthorization:*

```
public override void OnAuthorization(HttpActionContext actionContext)
{
	base.OnAuthorization(actionContext);
	string matricola = actionContext.RequestContext.Principal.Identity.Name;
	if (matricola == "")
	{
		// utente non autenticato
		actionContext.Response = new
			HttpResponseMessage(System.Net.HttpStatusCode.Unauthorized);
		actionContext.Response.ReasonPhrase = "Non si è autenticati";
		return;
	}
	using (ServerProgettoContext db = new ServerProgettoContext())
	{
		UtenteDB utenteDB = db.Utenti
			.Include(u => u.UtenteProfiliList.Select(p =>
				p.Profilo.ProfiloPermissionsList.Select(pp =>
			pp.Permission)))
			.FirstOrDefault(x => x.Matricola == matricola);
		if (utenteDB == null)
		{
			// non si dispone del permesso per utilizzare l'applicazione
			actionContext.Response = new
				HttpResponseMessage(System.Net.HttpStatusCode.Forbidden);
			actionContext.Response.ReasonPhrase = "Non si possiede
				l'autorizzazione per utilizzare l'applicazione";
			return;
		}
		IEnumerable<PermissionDB> permissionUtenteList =
			utenteDB.UtenteProfiliList.ElementAt(0)
			.Profilo.ProfiloPermissionsList.Select(pp =>
pp.Permission).Where(x
				=> x.CodPermission == _permissionName);
		if (permissionUtenteList.Count() == 0)
		{
			// non si dispone del permesso per l'operazione richiesta
			actionContext.Response = new
				HttpResponseMessage(System.Net.HttpStatusCode.Forbidden);
			actionContext.Response.ReasonPhrase = "Non si possiede
				l'autorizzazione per l'operazione richiesta";
			return;
		}
	}
}
```

5.5 Configurazione Entity Framework

Come è stato già anticipato nel capitolo precedente si è fatto utilizzo dell'approccio Code-First messo a disposizione dal framework. La classe context elenca tutte le classi ModelDB che vengono utilizzate per la costruzione delle tabelle del database:

```
public class ServerProgettoContext: DbContext, IServerProgettoContext
{
	public ServerProgettoContext() : base("name=ProgettoDBConnectionString")
	{
		Database.SetInitializer(new
MigrateDatabaseToLatestVersion<ServerProgettoContext, Migrations.Configuration>());
		Database.Initialize(false);
	}
	public DbSet<UtenteDB> Utenti { get; set; }
	public DbSet<PermissionDB> Permission { get; set; }
	public DbSet<VoceMenuDB> VociMenu { get; set; }
	public DbSet<MenuPermissionDB> MenuPermission { get; set; }
	public DbSet<ProfiloPermissionDB> ProfiloPermission { get; set; }
	public DbSet<UtenteProfiloDB> UtenteProfili { get; set; }
	public DbSet<CommessaDB> Commesse { get; set; }
	public DbSet<Macro_AttivitaDB> MacroAttivita { get; set; }
	public DbSet<ProfiloDB> Profili { get; set; }
	public DbSet<RisorsaDB> RisorseDB { get; set; }
	public DbSet<ProgettoDB> Progetti { get; set; }
	public DbSet<ProgettoTaskDB> ProgettoTasks { get; set; }
	public DbSet<ProgettoRisorsaDB> ProgettoRisorse { get; set; }
	public DbSet<RisorsaConsuntivazioneDB> RisorsaConsuntivazioni { get; set; }
	public DbSet<StatoAttivitaDB> StatiAttività { get; set; }
	public DbSet<StatoConsuntivoDB> StatiConsuntivi { get; set; }
	public DbSet<TipoRisorsaDB> TipoRisorseDB { get; set; }
	public DbSet<LogDB> Log { get; set; }

	protected override void OnModelCreating(DbModelBuilder modelBuilder)
	{ /*…*/ }

}
```

Come è possibile leggere dal codice soprastante le classi ModelDB sono in numero maggiore rispetto alle classi di dominio, perché all'interno del database devono essere salvate anche tutte le informazioni relative agli utenti e quelle relative alle voci di menu e alle permission associate ad un certo profilo. Inoltre, è stato utilizzato il tool Log4Net per loggare tutti gli errori che si verificano durante l'esecuzione del web service all'interno della tabella generata a partire dalla classe LogDB.

Il metodo *OnModelCreting* permette di definire delle convenzioni personalizzate che vengono utilizzate durante la costruzione delle tabelle del database, come per esempio la precisione di una proprietà decimal che viene mappata in una colonna di una tabella.

Il costruttore della classe context setta MigrateDatabaseToLatestVersion come Initializer del database. Questo Initializer è associato al tool di migrazione introdotto dall'Entity Framework che aggiorna in modo automatico lo schema del database ad ogni cambiamento del modello senza perdere i dati già presenti all'interno del DB. La configurazione delle migrazioni è delegata alla classe Configuration all'interno della directory Migrations.

5.6 Implementazione delle varie funzionalità

Nel corso di questo paragrafo verrà descritta l'implementazione delle funzionalità offerte dall'applicazione, in accordo con i requisiti che sono già stati discussi nel terzo capitolo. Tutte le griglie, ad eccezione di quella visualizzata nella Dashboard, che verranno mostrate durante la descrizione delle varie funzionalità sono componenti inclusi nella libreria Kendo UI for Angular. Utilizzando le tabelle di questa libreria è molto più semplice rendere le colonne filtrabili e ordinabili. Le opzioni di filtraggio sono molte e sono tutte rese disponibili all'utilizzatore dell'applicazione. Inoltre, con l'utilizzo di queste griglie è molto semplice anche implementare la funzionalità di export dei dati in formato Excel.

La vista di ogni tabella Kendo è modellata da un oggetto di tipo *Observable<GridDataResult>*, dove GridDataResult è un tipo importato dalla libreria Angular. L'utilizzo di un Observable fa sì che il popolamento della tabella sia asincrono. Infatti, tutti i metodi implementati nei service utilizzati per contattare le API del web service ritornano degli Observable. In questo modo, i componenti che chiamano questi metodi possono effettuare il subscribe con una funzione da eseguire nel momento in cui il download dei dati è terminato: tramite questa funzione è possibile aggiornare la lista dei dati che sono visualizzati nella griglia.

5.6.1 Dashboard

Questa pagina può essere visualizzata da qualsiasi utente autenticato e autorizzato ad utilizzare l'applicazione, a prescindere da quale sia il suo ruolo, in modo da conoscere l'andamento generale delle commesse, dei progetti e dei singoli task. La tabella che contiene queste informazioni ha una logica dietro di sé abbastanza complessa: infatti, viene costruita dinamicamente sfruttando le direttive e i binding di Angular e senza l'ausilio di alcuna libreria esterna. Ogni riga che contiene una commessa può essere espansa in modo da mostrarne i progetti e ogni riga che contiene un progetto può essere a sua volta espansa per mostrarne i task. La "Figura 17" mostra la schermata della Dashboard.

Il modulo Angular responsabile della gestione di questa funzionalità è il DashboardModule. Durante l'inizializzazione del suo unico componente, il DashboardComponent, viene utilizzato il service DashboardService per scaricare tutti i progetti (anche quelli completi) con data di inizio nel range di date che si estende per default dalla data corrente ai sei mesi precedenti. Per poter visualizzare all'interno della tabella anche i progetti ultimati basta agire sullo switch: internamente vengono infatti mantenute due liste, una con tutti i progetti e una solo con i progetti non ancora ultimati, in modo da aggiornare il contenuto della tabella con una di queste due liste in base al valore dello switch. Ogni qual volta viene modificato il range di date e si clicca sul pulsante FILTRA, vengono scaricati dal server i progetti che soddisfano i nuovi parametri. Il metodo esposto dal DashboardService incaricato di compiere questo compito è il seguente:

dashboard.service.ts

```
public fetch(Inizio: Date, Fine: Date): Observable<any[]> {
  return this.http.get<any[]>(this.dashboardUrl + "?inizio=" + Inizio.getTime()
    + "&fine=" + Fine.getTime());
```

Gli argomenti *Inizio* e *Fine* vengono settati dal DashboardComponent con il valore dei due DatePicker posti al di sopra della tabella.

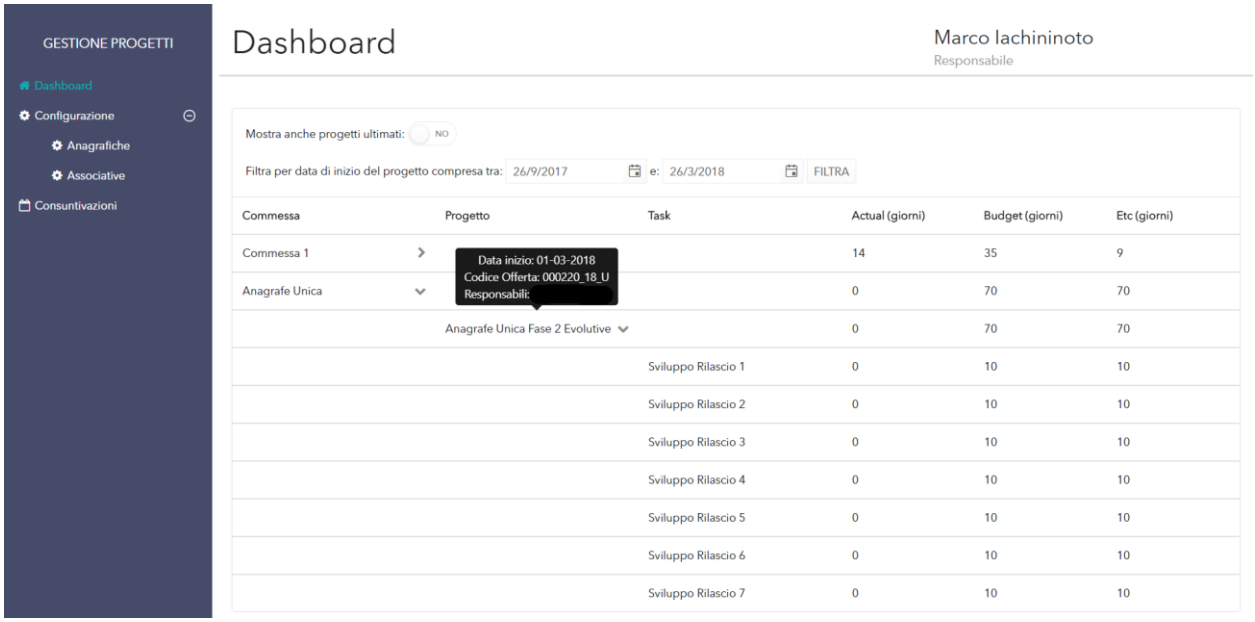


***Figura 17**: Dashboard*

Lato server la richiesta viene ricevuta dal controller DashboardController. Questo estrapola il valore delle due date ricevute come query parameter e chiama in causa il service ProgettoService. Viene utilizzato questo service perché per ottenere le informazioni da visualizzare nella Dashboard è necessario popolare tutti gli oggetti del dominio al di sotto dell'Aggregate Root Progetto per ogni progetto che soddisfa la query. Il ProgettoService incarica il ProgettoRepository ricevuto come dipendenza per ottenere le informazioni richieste dopo aver popolato gli oggetti del dominio.

Questo è uno dei pochissimi casi in cui una richiesta di lettura viene soddisfatta non utilizzando un oggetto Query Handler, ma utilizzando un Repository: le informazioni da leggere riguardo gli Etc, gli Actual e i Budget possono essere ottenute solo applicando le regole di business agli oggetti del dominio. Infatti, dopo aver popolato il dominio vengono calcolati Actual e Etc per i singoli task, utilizzando i metodi *TotaleConsuntivazioni* e *GetGiorniMancantiEtc* della classe ProgettoTask. Successivamente il calcolo viene effettuato a livello di Progetto tramite i metodi *GetTotaleConsuntivazioni* e *GetGiorniMancantiEtc* della classe Progetto. Le informazioni sul Budget sono mantenute come proprietà sia a livello di task che di Progetto e quindi non serve chiamare alcun metodo per ricavare questa informazione. Infine, lo stesso calcolo avviene a livello di Commessa, chiamando i metodi *GetTotaleConsuntivazioni*, *GetGiorniMancantiEtc* e *GetTotaleBudget*.

I dati così ottenuti sono modellati da una lista di CommesseForDashboardAPI. Ognuno di questi elementi della lista contiene un elenco di ProgettiForDashboardAPI e ognuno di questi progetti contiene a sua volta una lista di TaskForDashboardAPI. Questi tre tipi di oggetti rientrano tutti nel concetto di ModelAPI, perché non hanno nulla a che fare con le interfacce

IQuery e *IQueryResult* che vengono usate per modellare gli oggetti gestiti tramite Query Handler.

5.6.2 Anagrafiche

Nell'applicazione Angular, il modulo AnagraficheModule si occupa della gestione delle funzionalità relative alle anagrafiche. Per gestire il routing all'interno di questo modulo è stato utilizzato un altro modulo che ha il nome di AnagraficheRoutingModule.

Tramite un tab posto nella parte superiore della pagina è possibile navigare tra le anagrafiche delle commesse, dei progetti e delle risorse. La costruzione delle voci di menu avviene in modo dinamico usando le direttive e il data-binding di Angular ed è a carico del component anagrafiche.component: questo è il component principale dell'AnagraficheModule. Di default l'utente è indirizzato nella pagina relativa ai progetti.

5.6.2.1 Commesse

Cliccando sul tab relativo alle commesse si ha la possibilità di gestire l'anagrafica delle commesse. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica delle commesse.

***Tabella 14**: Componenti front-end*

	Component	Service	Model
Visualizzazione	anagrafiche-commesse.component	anagrafiche-commesse.service	
		anagrafiche-commesse.service	
Inserimento/modifica	edit-commesse-form.component	anagrafiche-commesse.service	modelResponsabile.ts
			model.ts (modella una commessa)

***Tabella 15:** Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	CommesseController: Get	SM_ANAG_COMM_R	ICommessaService	GetCommesseForAdminQueryHandler.cs
				GetCommesseQueryHandler.cs
Inserimento	CommesseController: Post	ANAG_COMM_ADD	ICommessaService	ICommessaRepository
Modifica	CommesseController: Put	SM_ANAG_COMM_W	ICommessaService	ICommessaRepository

Il componente anagrafiche-commesse.component, durante la sua inizializzazione, invoca il metodo *getCommesse* del service anagrafiche-commesse.service, in modo da scaricare l'elenco delle commesse. Lato server, il controller chiama in causa il service ICommessaService che usa IRolesService per decidere se l'utente corrente è un responsabile, caso in cui la lettura delle commesse è delegata al GetCommesseQueryHandler.cs, oppure un admin, caso in cui la lettura è delegata al GetCommesseForAdminQuery. Nel primo caso verranno scaricate solo le commesse di cui l'utente è responsabile.

Per poter visualizzare il bottone "Aggiungi Commessa", l'utente corrente deve possedere il permesso "ANAG_COMM_ADD". Questo controllo è effettuato invocando il metodo *GetUser* dell'authentication.service che ritorna tutte le informazioni sull'utente (quindi anche i permessi). La "Figura 18" mostra un esempio di griglia con all'interno due commesse.

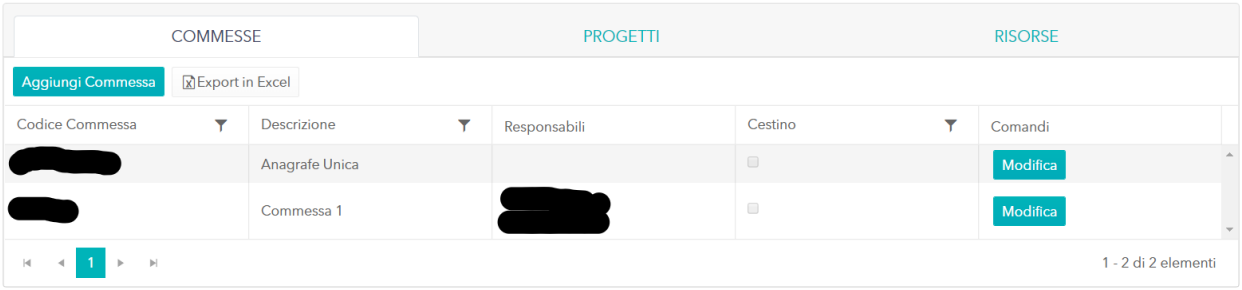


***Figura 18**: Tabella commesse*

Quando si preme il bottone "Aggiungi Commessa" o "Modifica" appare un pop-up, come mostrato nella "Figura 19". Nel caso della modifica i campi del form sono preimpostati con gli attributi della commessa che si vuole modificare. Il metodo *getRisorse* dell'anagrafiche-commesse-service viene utilizzato per scaricare quelle risorse (modellate da modelResponsabile) associate ad un'utenza amministratore o responsabile e che quindi possono essere selezionate come responsabili della commessa. Questa richiesta viene soddisfatta dall'action method *GetRisorseForDropdownInCommessa* del controller RisorseQueryController, che per ogni risorsa ritorna solo il suo ID e il suo nome completo (nome più cognome). Il campo relativo ai responsabili è modellato tramite il componente Kendo Multiselect. Alla pressione del pulsante "Salva" viene chiamato il metodo *addCommessa*, nel caso di inserimento, oppure *updateCommessa*: entrambi sono metodi dell'anagrafiche-commesse-service. Lato server, sia in caso di aggiunta che di modifica, ICommessaRepository applicherà direttamente le modifiche al database tramite l'Entity Framework senza costruire e popolare alcun oggetto di dominio perché non ci sono regole di business da verificare.

Se l'utente ha il profilo Responsabile, non basta solo il permesso SM_ANAG_COMM_W per determinare se può modificare la commessa: la modifica è permessa solo se l'utente è proprio il responsabile della commessa che si vuole modificare; se è amministratore non ha alcuna restrizione. Questo ulteriore controllo è effettuato lato server dal service ICommessaService sfruttando il service di supporto IRolesService.

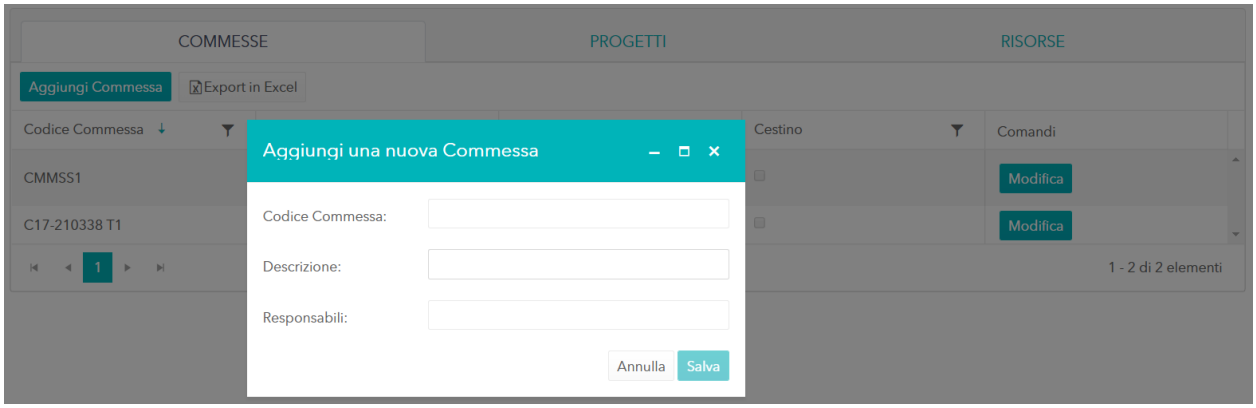


Figura 19: Inserimento commessa

5.6.2.2 Progetti

Cliccando sul tab Progetti si ha la possibilità di gestire l'anagrafica dei progetti. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica dei progetti.

***Tabella 16**: Componenti front-emd*

	Component	Service	Model
Visualizzazione	anagrafiche-progetti.component	anagrafiche-progetti.service	
Inserimento/modifica	edit-progetti-form.component	anagrafiche-progetti.service	modelCommessa.ts
			model.ts (modella un progetto)

***Tabella 17**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	ProgettiController: Get	SM_ANAG_PROG_R	IProgettoService	GetProgettiForAdminQueryHandler.cs
				GetProgettiQueryHandler.cs
Inserimento	ProgettiController: Post	SM_ANAG_PROG_W	IProgettoService	IProgettoRepository
Modifica	ProgettiController: Put	SM_ANAG_PROG_W	IProgettoService	IProgettoRepository

Il componente anagrafiche-progetti.component, mentre viene inizializzato, invoca il metodo *GetProgetti* del service anagrafiche-progetti.service, così da ottenere l'elenco dei progetti. Nel back-end, il service chiamato in causa dal controller utilizza a sua volta il service IRolesService per decidere se l'utente corrente è un responsabile, caso in cui la lettura dei progetti è delegata al GetProgettiQueryHandler.cs, oppure un admin, caso in cui la lettura è delegata al

GetProgettiForAdminQuery. Nel primo caso verranno scaricate solo i progetti di cui l'utente è responsabile o che appartengono ad una commessa di cui l'utente è responsabile commessa. La "Figura 20" mostra un esempio di griglia contenente tre progetti.

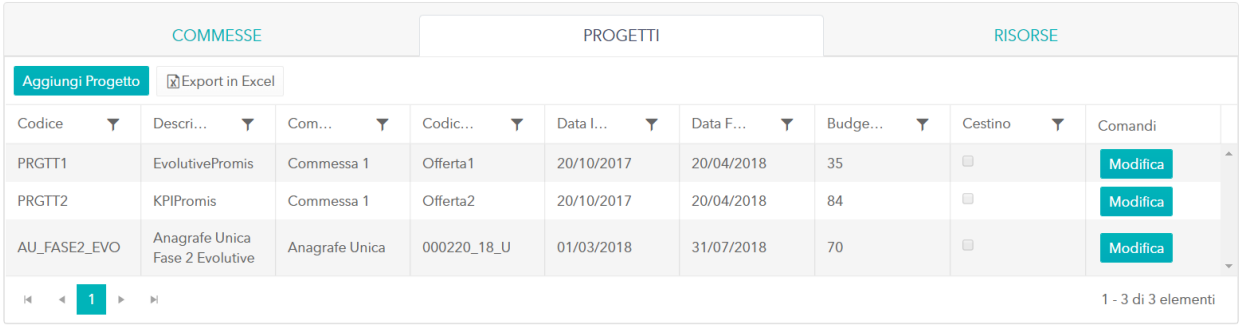


***Figura 20**: Griglia progetti*

Premendo il bottone "Aggiungi Progetto" o "Modifica" appare un pop-up, come mostrato nella "Figura 21". Se il bottone premuto è "Modifica" i campi del form sono preimpostati con gli attributi del progetto che si sta modificando. Il metodo *getCommesse* dell'anagrafiche-progetti.service scarica tutte le commesse disponibili (modellate dal modelCommessa.ts) in modo da popolare la sorgente dati del campo Commessa, modellato dal componente Kendo Combobox. Questa richiesta viene soddisfatta dall'action method *GetCommesseForDropdown* del controller CommesseQueryController. Alla pressione del pulsante "Salva" viene chiamato il metodo *addProgetto*, nel caso di inserimento, oppure *updateProgetto*: entrambi sono forniti dall'anagrafiche-progetti.service. Lato server, sia in caso di aggiunta che di modifica, IProgettoRepository costruisce e popola la commessa (oggetto del dominio) a cui fa riferimento il Progetto e controlla se l'operazione richiesta non solleva alcuna eccezione. In questo caso il dato viene reso persistente tramite l'utilizzo dell'Entity Framework.

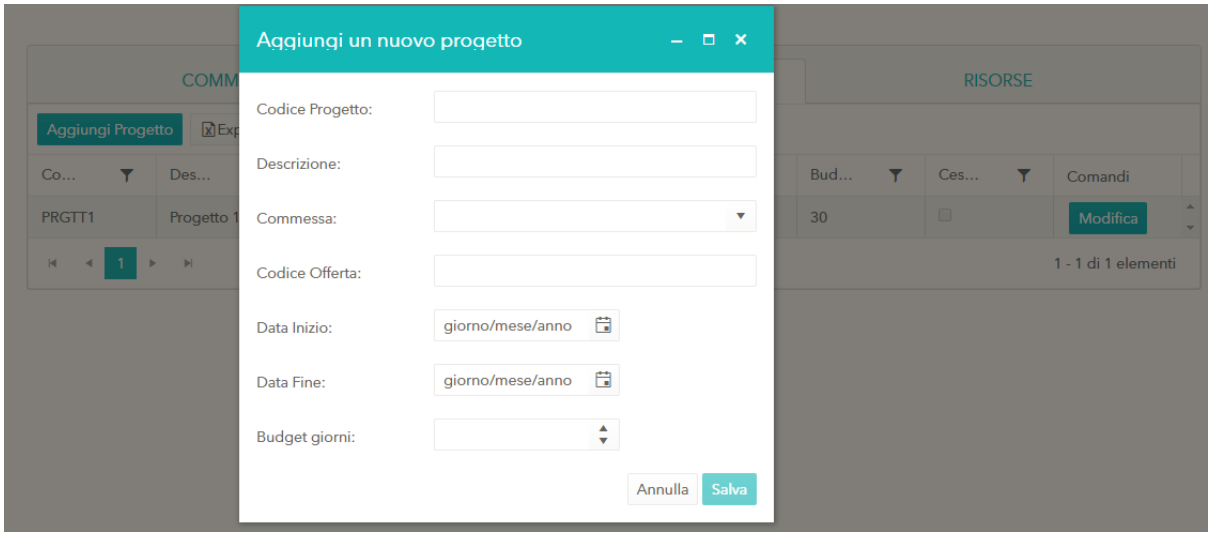


***Figura 21**: Inserimento progetto*

Nel caso di inserimento non basta possedere il permesso SM_ANAG_PROG_W: infatti se l'utente ha come profilo di accesso all'applicazione quello di Responsabile può inserire solo se è il responsabile della commessa a cui il progetto fa riferimento. Stessa cosa accade nel caso di modifica: se l'utente è Responsabile, può effettuare l'operazione solo se è responsabile del progetto o della commessa associata al progetto su cui si sta agendo. Questi ulteriori controlli sono effettuati all'interno dell'IProgettoService facendo utilizzo di IRolesService.

5.6.2.3 Risorse

Facendo click sul tab Risorse si può gestire l'anagrafica delle risorse. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica delle risorse.

***Tabella 18**: Componenti front-emd*

	Component	Service	Model
Visualizzazione	anagrafiche-risorse.component	anagrafiche-risorse.service	
Inserimento/modifica	edit-risorse-form.component	anagrafiche-risorse.service	modelMatricolaUtente.ts
			modelTipoRisorsa.ts
			model.ts (modella una risorsa)

***Tabella 19**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	RisorseController: Get	SM_ANAG_RIS_R	IRisorsaService	GetRisorseQueryHandler.cs
Inserimento	RisorseController: Post	SM_ANAG_RIS_W	IRisorsaService	IRisorsaRepository
Modifica	RisorseController: Put	SM_ANAG_RIS_W	IRisorsaService	IRisorsaRepository

Il componente anagrafiche-risorse.component invoca il metodo *GetRisorse* del service anagrafiche-risorse.service durante la sua inizializzazione, così da ottenere l'elenco delle risorse. La "Figura 22" mostra un esempio di griglia con cinque risorse.

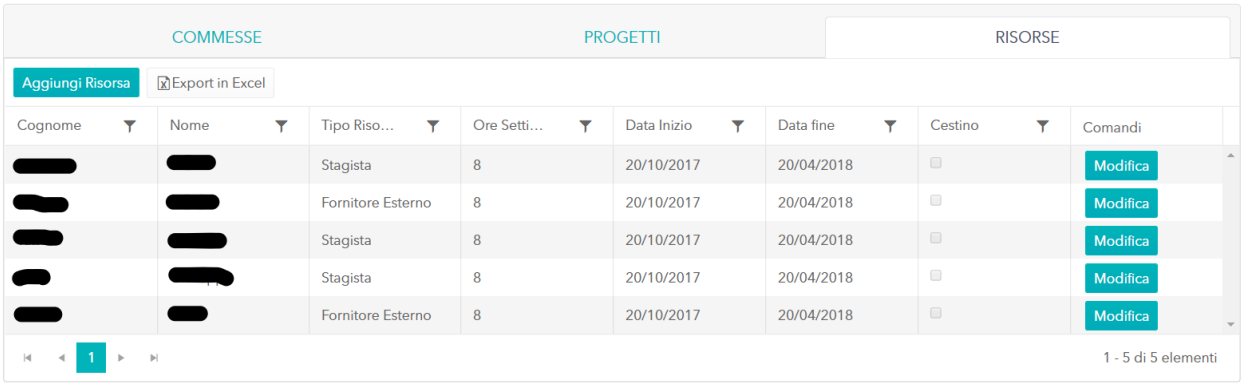


***Figura 22**: Griglia risorse*

Facendo click sul bottone "Aggiungi Risorsa" o "Modifica" appare un pop-up, come mostrato nella "Figura 23". In caso di modifica i campi del form sono preimpostati con il valore della risorsa che è stata selezionata. Il metodo *getTipiRisorsa* dell'anagrafiche-risorse.service scarica i tipi risorsa disponibili per il campo TipoRisorsa, modellato dal componente Kendo Combobox. Questa richiesta è indirizzata all'action method *GetTipiRisorseForDropdown* del controller TipiRisorseQueryController. Il metodo *getMatricole* dell'anagrafiche-risorse.service scarica le matricole disponibili per il campo Matricola, modellato dal componente Kendo Autocomplete. Lato server, questa richiesta è soddisfatta dall'action method *GetMatricoleUtenti* del controller MatricoleUtentiController. Alla pressione del pulsante "Salva" viene chiamato il metodo *addRisorsa*, nel caso di inserimento, oppure *updateRisorsa*: entrambi sono forniti dall'anagrafiche-risorse.service. Lato server, sia in caso di aggiunta che di modifica, IRisorsaRepository renderà direttamente i dati persistenti tramite l'Entity Framework senza costruire e popolare alcun oggetto di dominio perché non ci sono regole di business da verificare.

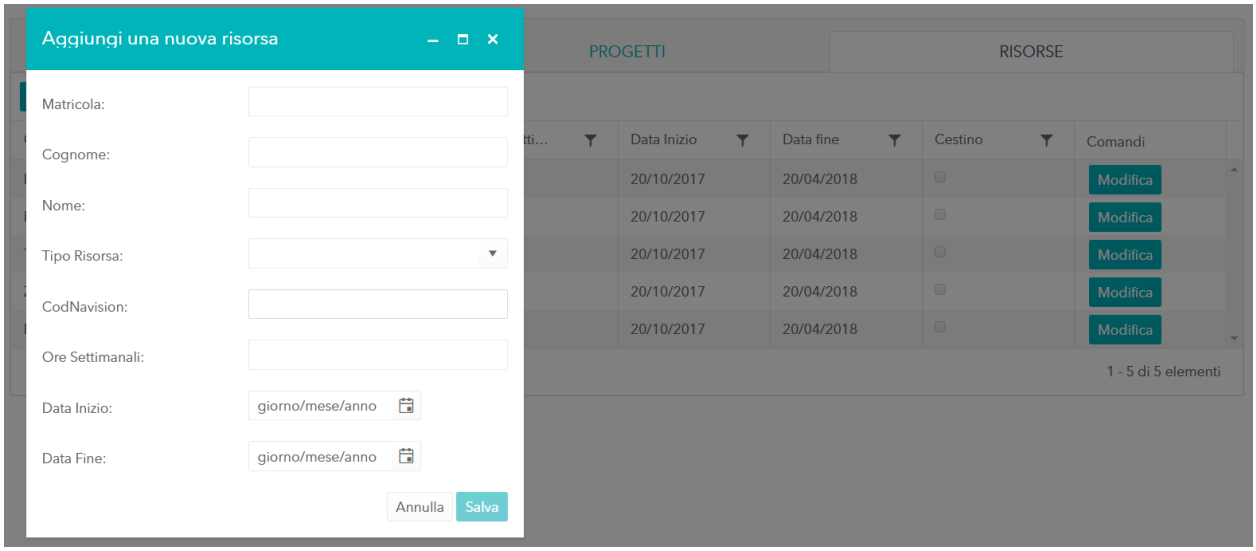


***Figura 23**: Inserimento risorsa*

5.6.3 Tabelle dominio

Nell'applicazione Angular, il modulo TabelleDominioModule contiene le funzionalità relative alle tabelle di dominio. Per gestire il routing all'interno di questo modulo è stato utilizzato un altro modulo che ha il nome di TabelleDominioRoutingModule.

Tramite un tab posto nella parte superiore della pagina è possibile navigare tra le varie tabelle di dominio: Profili, Tipo Risorse, Stati Consuntivi e Macro-Attività. La costruzione delle voci di menu è dinamica ed è effettuata usando le direttive e il data-binding di Angular. Questa operazione è svolta dal component tabelle-dominio.component, che è il component principale del modulo TabelleDominioModule. Di default l'utente è indirizzato nella pagina relativa ai profili.

Tutti i repository coinvolti nelle operazioni di inserimento o modifica delle tabelle di dominio agiscono direttamente sul database tramite l'Entity Framework. Infatti, non ci sono regole di business da verificare e quindi non serve creare e popolare alcun oggetto di dominio.

5.6.3.1 Profili

Cliccando sul tab relativo ai profili si accede alla pagina che permette di gestire i profili. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica dei profili.

***Tabella 20**: Componenti front-end*

	Component	Service	Model
Visualizzazione	profili.component	profili.service	
Inserimento/modifica	edit-profili-form.component	profili.service	model.ts (modella un profilo)

***Tabella 21**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	ProfiliController: Get	SM_TAB_DOM_RW	IProfiloService	GetProfiloQueryHandler.cs
Inserimento	ProfiliController: Post	SM_TAB_DOM_RW	IProfiloService	IProfiloRepository
Modifica	ProfiliController: Put	SM_TAB_DOM_RW	IProfiloService	IProfiloRepository

Il componente profili.component invoca il metodo *GetProfili* del service profili.service quando viene inizializzato e scarica l'elenco dei profili. La "Figura 24" mostra la griglia con i tre profili espressi nei requisiti.

***Figura 24**: Tabella profili*

Cliccando sul pulsante "Aggiungi Profilo" o "Modifica" viene mostrato un pop-up, come nella "Figura 25". In caso di modifica i campi del form sono preimpostati con il valore del profilo selezionato. Premendo il pulsante "Salva" nel caso di inserimento si chiama il metodo *addProfilo* del service profili.service, mentre nel caso di modifica si chiama il metodo *updateProfilo* sempre dello stesso service.

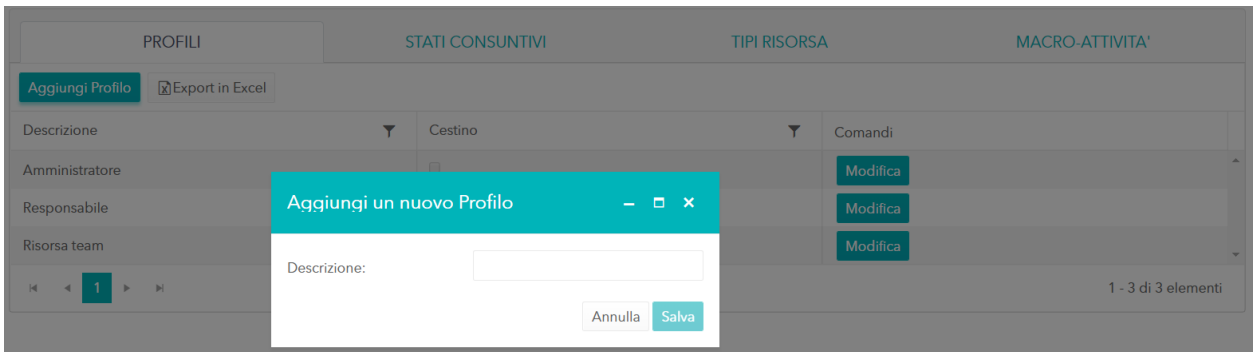


***Figura 25**: Inserimento profilo*

5.6.3.2 Stati consuntivi

Cliccando sul tab Stati Consunitivi è possibile gestire la tabella di dominio degli Stati Consuntivi. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica degli stati consuntivi.

***Tabella 22**: Componenti front-end*

	Component	Service	Model
Visualizzazione	stati-consuntivi.component	stati-consuntivi.service	
Inserimento/modifica	edit-stati-consuntivi-form.component	stati-consuntivi.service	model.ts (modella uno stato consuntivo)

***Tabella 23**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	StatiConsuntiviController: Get	SM_TAB_DOM_RW	IStatoConsuntivoService	GetStatiConsuntiviQueryHandler.cs
Inserimento	StatiConsuntiviController: Post	SM_TAB_DOM_RW	IStatoConsuntivoService	IStatoConsuntivoRepository
Modifica	StatiConsuntiviController: Put	SM_TAB_DOM_RW	IStatoConsuntivoService	IStatoConsuntivoRepository

Il componente stati-consuntivi.component invoca il metodo *GetStatiConsuntivi* del service stati-consuntivi.service durante la sua inizializzazione e scarica l'elenco degli stati consuntivi. La "Figura 26" mostra la griglia con gli stati consuntivi indicati dai requisiti.

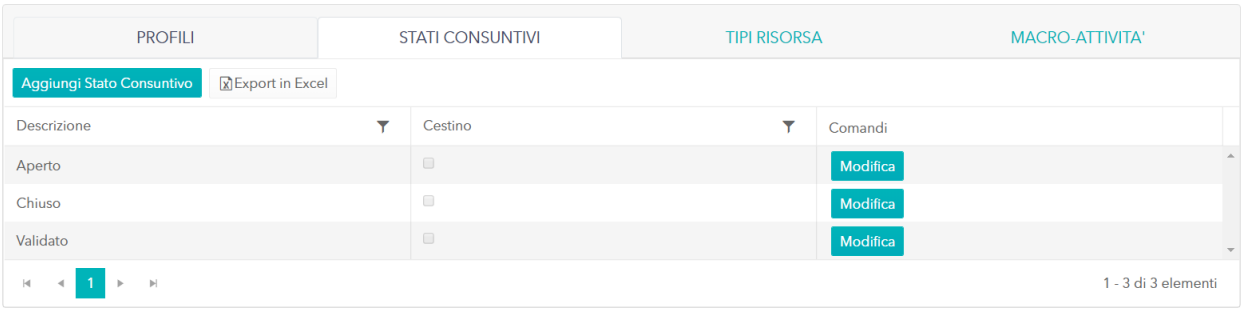


***Figura 26**: Griglia stati consuntivi*

Cliccando sul pulsante "Aggiungi Stato Consuntivo" o "Modifica" appare un pop-up, come nella "Figura 27". In caso di modifica i campi del form sono preimpostati con il valore dello stato consuntivo selezionato. Premendo il pulsante "Salva" nel caso di inserimento si chiama il metodo *addStatoConsuntivo* del service stati-consuntivi.service, mentre nel caso di modifica viene chiamato il metodo *updateStatoConsuntivi* sempre dello stesso service.

***Figura 27**: Inserimento stato consuntivo*

5.6.3.3 *Tipi risorsa*

Cliccando sul tab Tipi risorsa è possibile gestire la tabella di dominio dei Tipi Risorsa. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica dei tipi risorsa.

***Tabella 24**: Componenti front-end*

	Component	Service	Model
Visualizzazione	tipi-risorsa.component	tipi-risorsa.service	
Inserimento/modifica	edit-tipi-risorsa-form.component	tipi-risorsa.service	model.ts (modella un tipo risorsa)

***Tabella 25**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	TipiRisorsaController: Get	SM_TAB_DOM_RW	ITipoRisorsaService	GetTipiRisorsaQueryHandler.cs
Inserimento	TipiRisorsaController: Post	SM_TAB_DOM_RW	ITipoRisorsaService	ITipoRisorsaRepository
Modifica	TipiRisorsaController: Put	SM_TAB_DOM_RW	ITipoRisorsaService	ITipoRisorsaRepository

Il componente tipi-risorsa.component durante la sua inizializzazione scarica l'elenco dei tipi risorsa invocando il metodo *GetTipiRisorse* del service tipi-risorsa.service. La "Figura 28" mostra la griglia con i tipi risorsa indicati dai requisiti.

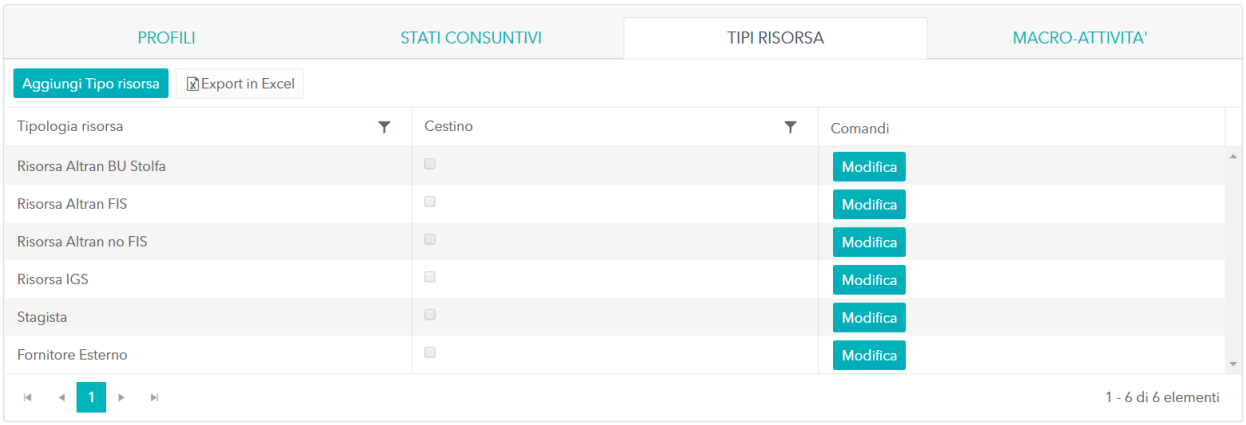


***Figura 28**: Tabella tipi risorsa*

Cliccando sul pulsante "Aggiungi Tipo Risorsa" o "Modifica" si apre un pop-up, come nella "Figura 29". In caso di modifica i campi del form sono preimpostati con il valore del tipo risorsa. Premendo il pulsante "Salva" nel caso di inserimento si chiama il metodo

addTipoRisorsa del service tipi-risorsa.service, mentre nel caso di modifica viene chiamato il metodo *updateTipoRisorsa* sempre dello stesso service.

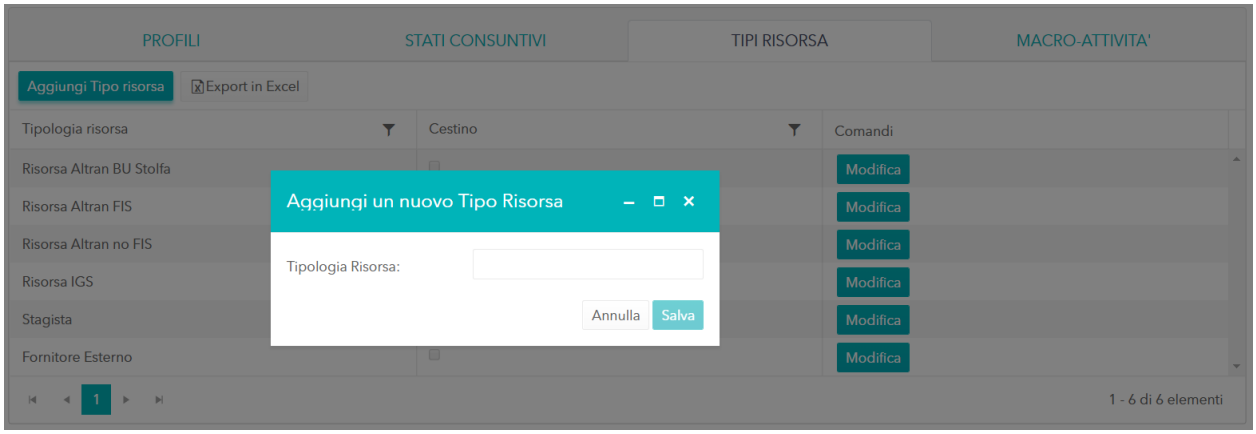


***Figura 29**: Inserimento tipo risorsa*

5.6.3.4 Macro-Attività

Facendo click sul tab Macro-Attività è possibile gestire la tabella di dominio delle Macro-Attività. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica delle macro-attività.

***Tabella 26**: Componenti front-end*

	Component	Service	Model
Visualizzazione	macro-attivita.component	macro-attivita.service	
Inserimento/modifica	edit-macro-attivita-form.component	macro-attivita.service	model.ts (modella una macro-attività)

***Tabella 27**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	MacroAttivitaController: Get	SM_TAB_DOM_RW	IMacroAttivitaService	GetMacroAttivitaQueryHandler.cs
Inserimento	MacroAttivitaController: Post	SM_TAB_DOM_RW	IMacroAttivitaService	IMacroAttivitaRepository
Modifica	MacroAttivitaController: Put	SM_TAB_DOM_RW	IMacroAttivitaService	IMacroAttivitaRepository

Il componente macro-attivita.component durante la sua inizializzazione chiama il metodo *GetMacroAttivita* del service macro-attività.service per scaricare le macro attività. La "Figura 30" mostra la griglia con le macro-attivita espresse nei requisiti.

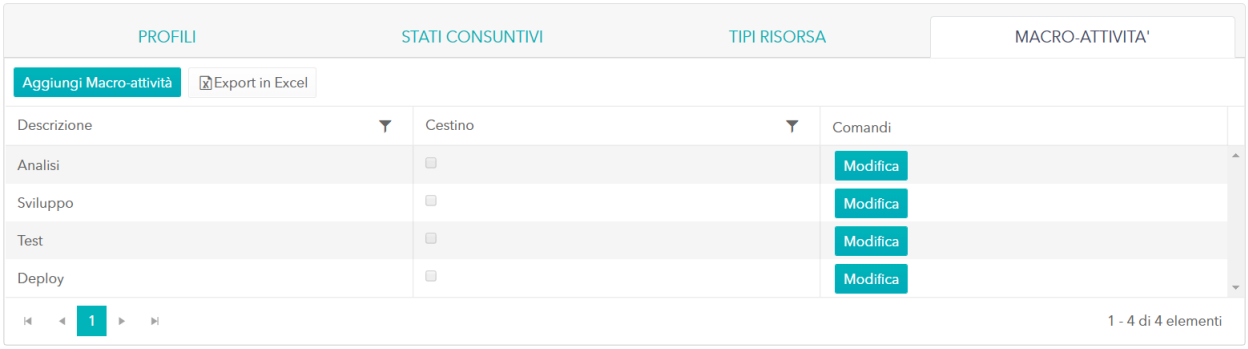


***Figura 30**: Tabella macro-attività*

Cliccando sul pulsante "Aggiungi Macro Attivita" o "Modifica" viene mostrato un pop-up, come nella "Figura 31". In caso di modifica i campi del form sono preimpostati con il valore della macro-attività scelta. Alla pressione del pulsante "Salva" viene chiamato il metodo *addMacroAttivita*, nel caso di inserimento, oppure *updateMacroAttivita*: entrambi sono forniti dal macro-attivita.service.

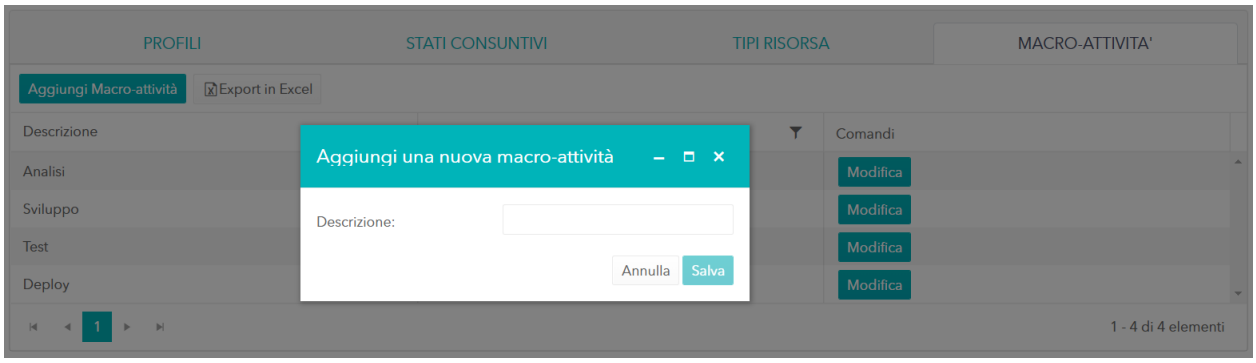


***Figura 31**: Inserimento macro-attività*

5.6.4 Associative

Il modulo AssociativeModule dell'applicazione Angular si occupa della gestione delle funzionalità relative alle associative. Per gestire il routing all'interno di questo modulo è stato utilizzato il modulo con nome AssociativeRoutingModule.

Tramite un tab posto nella parte superiore della pagina è possibile navigare tra le associative Progetto-Risorsa e Progetto-Task. La costruzione delle voci di menu è dinamica ed è effettuata dal component associative.component, che è il component principale del modulo AssociativeModule. Di default l'utente è indirizzato nella pagina relativa ai Progetto-Risorsa.

5.6.4.1 Progetto-Risorsa

Facendo click sul tab Progetto-Risorsa viene mostrata la pagina per la gestione dell'associazione tra progetti e risorse. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica dei progetto-risorse.

***Tabella 28**: Componenti front-end*

	Component	Service	Model
Visualizzazione	associative-progetto-risorse.component	associative-progetto-risorse.service	
Inserimento/modifica	edit-progetto-risorse-form.component	associative-progetto-risorse.service	modelProgettoRisorsa.ts
			modelProgetto.ts
			modelRisorsa.ts
			modelProfilo.ts

***Tabella 29**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	ProgettoRisorseController: Get	SM_ASS_PROG_RIS_R	IProgettoRisorsaService	GetProgettoRisorseQueryHandler
Inserimento	ProgettoRisorseController: Post	SM_ASS_PROG_RIS_W	IProgettoRisorsaService	IProgettoRepository
Modifica	ProgettoRisorseController: Put	SM_ASS_PROG_RIS_W	IProgettoRisorsaService	IProgettoRepository

Tramite una Kendo Combobox l'utente seleziona il progetto di interesse. Questi progetti (modellati da modelProgetto.ts) sono scaricati durante l'inizializzazione dell'associative-progetto-risorse.component chiamando il metodo *getProgetti* dell'associative-progetto-risorse.service. Lato server, questa richiesta viene gestita dall'action method *GetProgettiForDropdown* del ProgettoQueryController. Nel caso in cui l'utente sia associato al profilo Responsabile, verranno scaricati solo i progetti di cui esso è responsabile o che appartengono a commesse di cui è responsabile commessa. Questo controllo è effettuato all'interno di IProgettoService tramite il service IRolesService. Se l'utente è un amministratore allora la richiesta è soddisfatta dal GetProgettiForDropdownForAdminQueryHandler, mentre se è un responsabile viene utilizzato il GetProgettiForDropdownQuery. Dopo aver scelto il progetto, l'associative-progetto-risorse.component chiama il metodo *GetProgettoRisorse* dell'associative-progetto-risorse.service in modo da riempire la griglia dei progetti-risorse, come nella "Figura 32".

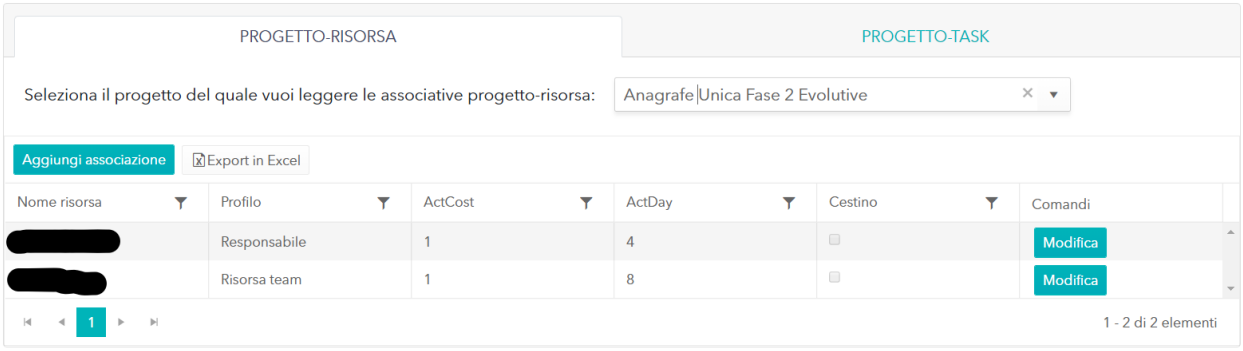


***Figura 32**: Griglia progetto-risorse*

Premendo il bottone "Aggiungi associazione" o "Modifica" appare un pop-up, come mostrato nella "Figura 33". Se il bottone premuto è "Modifica" i campi del form sono preimpostati con gli attributi del progetto-risorsa che si sta modificando. Il metodo *getProfili* dell'associative-progetto-risorse.service scarica i profili disponibili (modellati da modelProfilo.ts), ad eccezione di quello Amministratore, in modo da popolare la sorgente dati del campo Profilo, che è un componente Kendo Combobox. Questa richiesta viene soddisfatta dall'action method *GetProfiliForDropdown* del controller ProfiliQueryController. Questo controller chiama in causa IProfiloService che incarica il GetProfiliForDropdownQueryHandler di ottenere i profili, escluso quello di Amministratore. Contemporaneamente viene chiamato il metodo *getRisorse*, fornito sempre dall'associative-progetto-risorse.service per scaricare le risorse disponibili (modellate da modelRisorsa.ts) e popolare i dati della Combobox per il campo Risorsa. Lato server, la richiesta è gestita dall'action method *GetRisorseForDropdown* del controller RisorseQueryController, che a sua volta invoca IRisorsaService. La lettura delle risorse dal database è delegata al GetRisorseForDropdownQueryHandler. Alla pressione del pulsante "Salva" viene chiamato il metodo *addProgettoRisorsa*, nel caso di inserimento, oppure *updateProgettoRisorsa*: entrambi sono forniti dall'associative-progetto-risorse.service. Lato server, sia in caso di aggiunta che di modifica, viene chiamato in causa IProgettoRepository perché si sta agendo su un'entità che fa parte dell'Aggregate Root Progetto. Quindi, il repository costruisce e popola il progetto (oggetto del dominio) a cui fa riferimento il ProgettoRisorsa e controlla se l'operazione richiesta non generi alcuna eccezione. In questo caso il dato viene reso persistente tramite l'utilizzo dell'Entity Framework.

Sia nel caso di inserimento che di modifica non basta possedere il permesso SM_ASS_PROG_RIS_W se l'utente ha come profilo di accesso all'applicazione quello di Responsabile. Infatti, esso può effettuare l'operazione solo se è il responsabile della commessa a cui il progetto fa riferimento o se è responsabile del progetto stesso. Questi ulteriori controlli sono effettuati all'interno dell'IProgettoRisorsaService, facendo utilizzo di IRolesService, prima di chiamare in causa il repository.

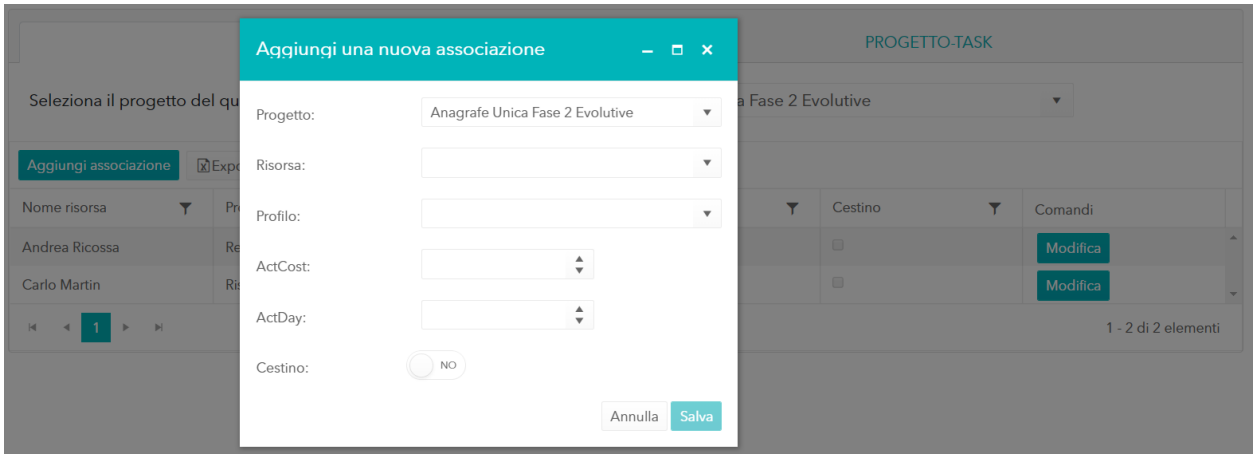


***Figura 33**: Inserimento Progetto-Risorsa*

5.6.4.2 Progetto-Task

Facendo click sul tab Progetto-Task si va alla pagina per la gestione dell'associazione tra progetti e task, con la possibilità di settare le risorse che lavoreranno ai task. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica dei progetto-task.

***Tabella 30**: Componenti front-end*

	Component	Service	Model
Visualizzazione	associative-progetto-task.component	associative-progetto-task.service	
Inserimento/modifica	edit-progetto-task-form.component	associative-progetto-task.service	modelProgettoRisorsa.ts
			modelProgettoWithBudget.ts
			modelMacroAttivita.ts
			modelProgettoTask.ts

***Tabella 31**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	ProgettoTaskController: Get	SM_ASS_PROG_TASK_R	IProgettoTaskService	GetProgettoTaskQueryHandler
Inserimento	ProgettoTaskController: Post	SM_ASS_PROG_TASK_W	IProgettoTaskService	IProgettoRepository
Modifica	ProgettoTaskController: Put	SM_ASS_PROG_TASK_W	IProgettoTaskService	IProgettoRepository

Tramite una Kendo Combobox l'utente seleziona il progetto di interesse. Questi progetti (modellati da modelProgettoWithBudget.ts) sono scaricati durante l'inizializzazione

dell'associative-progetto-task.component chiamando il metodo *getProgetti* dell'associative-progetto-task.service. Lato server, questa richiesta viene gestita dall'action method *GetProgettiWithBudget* del ProgettoQueryController. Queste informazioni permettono all'utente di conoscere il budget del Progetto, il budget già assegnato e quello ancora disponibile nel momento in cui farà un inserimento o una modifica. Nel caso in cui l'utente sia associato al profilo Responsabile, verranno scaricati solo i progetti di cui esso è responsabile o che appartengono a commesse di cui è responsabile commessa. Questo controllo è effettuato all'interno di IProgettoService tramite il service IRolesService. A differenza del caso Progetto-Risorsa, IProgettoService non fa utilizzo di un Query Handler per ottenere i progetti, ma chiama in causa IProgettoRepository. Il motivo è analogo a quello che è già stato spiegato nel caso della Dashboard. Infatti, per calcolare il budget già assegnato ad un progetto bisogna invocare il metodo *GetBudgetAssegnato* della classe di dominio Progetto. Questo vuol dire che bisogna costruire e popolare gli oggetti del dominio al di sotto dell'Aggregate Root Progetto. Dopo aver scelto il progetto, l'associative-progetto-task.component chiama il metodo *GetProgettoTasks* dell'associative-progetto-task.service in modo da riempire la griglia dei progetti-task, come nella "Figura 34".

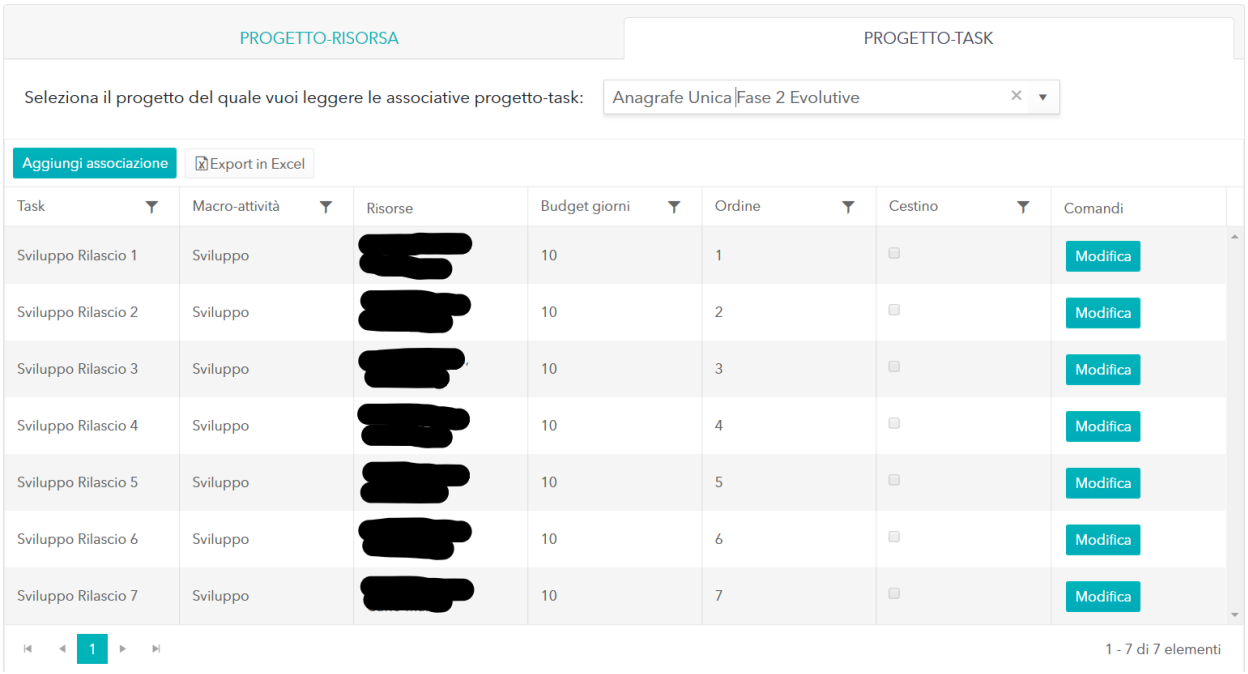


***Figura 34**: Tabella Progetto-Task*

Facendo click su "Aggiungi associazione" o "Modifica" appare un pop-up, come mostrato nella "Figura 35". Se il bottone premuto è "Modifica" i campi del form sono preimpostati con gli attributi del progetto-task che si sta modificando. Il metodo *getProgettoRisorse* dell'associative-progetto-task.service scarica le risorse già associate al progetto (modellati da modelProgettoRisorsa.ts), in modo da popolare la sorgente dati del campo Risorse, che è un componente Kendo Multiselect. Questa richiesta è gestita dall'action method *Get* del controller ProgettoRisorseQueryController. Questo controller incarica IProgettoRisorsaService che ottiene le informazioni dal database tramite il GetProgettoRisorsaForDropdownQueryHandler.

Contemporaneamente viene chiamato il metodo *getMacroAttività*, messo a disposizione sempre dall'associative-progetto-task.service per popolare la sorgente dati della Combobox associata al campo macro-attività. Lato server, la richiesta è gestita dall'action method *GetMacroForDropdown* del controller MacroAttivitaQueryController, che a sua volta invoca IMacroAttivitaService. La lettura delle macro-attività dal database è delegata al GetMacroForDropdownQueryHandler. Alla pressione del pulsante "Salva" viene chiamato il metodo *addProgettoTask*, nel caso di inserimento, oppure *updateProgettoTask*: entrambi sono forniti dall'associative-progetto-task.service. Lato server, sia in caso di aggiunta che di modifica, viene chiamato in causa IProgettoRepository perché si sta agendo su un'entità che fa parte dell'Aggregate Root Progetto. Quindi, il repository costruisce e popola il progetto (oggetto del dominio) a cui fa riferimento il ProgettoTask e tenta l'operazione richiesta invocando il metodo opportuno della classe Progetto. Se nessuna eccezione viene sollevata il dato viene reso persistente tramite l'utilizzo dell'Entity Framework.

Sia nel caso di inserimento che di modifica non basta possedere il permesso SM_ASS_PROG_RIS_W se l'utente ha come profilo di accesso all'applicazione quello di Responsabile. Infatti, esso può effettuare l'operazione solo se è il responsabile della commessa a cui il progetto fa riferimento o se è responsabile del progetto stesso. Questi ulteriori controlli sono effettuati all'interno dell'IProgettoTaskService, facendo utilizzo di IRolesService, prima di chiamare in causa il repository.

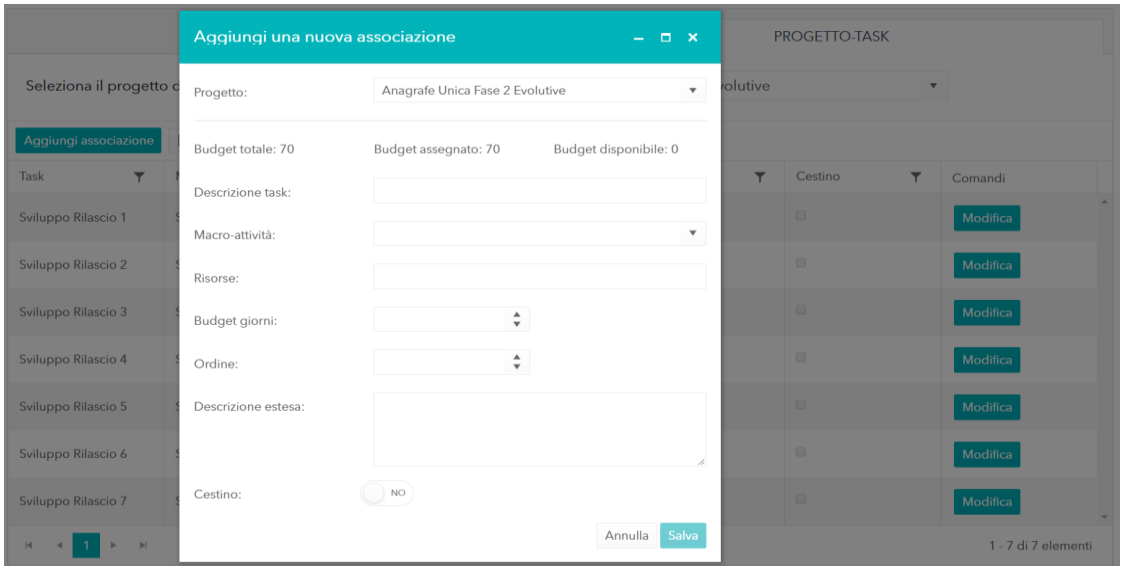


***Figura 35**: Inserimento Progetto-Task*

5.6.5 Consuntivazioni

Il modulo ConsuntivazioniModule è responsabile della gestione delle funzionalità relative alle consuntivazioni delle risorse. Le tabelle sottostanti elencano gli elementi front-end e back-end che vengono chiamati in causa durante le operazioni di visualizzazione e inserimento/modifica delle consuntivazioni.

***Tabella 32**: Componenti front-end*

	Component	Service	Model
Visualizzazione	consuntivazioni.component	consuntivazioni.service	
Inserimento/modifica	edit-consuntivazioni-form.component	consuntivazioni.service	modelConsuntivazione.ts
			modelProgetto.ts
			modelRisorsa.ts
			modelStato.ts
			modelTask.ts

***Tabella 33**: Componenti back-end*

	Controller: Action	Permesso	Service	Query Handler/Repository
Visualizzazione	ConsuntivazioniController: Get	MN_CONS_R	IConsuntivazioniService	GetConsuntivazioniQueryHandler
				GetConsuntivazioniForDevQueryHandler
Inserimento	ConsuntivazioniController: Post	MN_CONS_W	IConsuntivazioniService	IProgettoRepository
Modifica	ConsuntivazioniController: Put	MN_CONS_W	IConsuntivazioniService	IProgettoRepository

Una Kendo Combobox permette all'utente di selezionare il progetto di interesse. Questi progetti (modellati da modelProgetto.ts) sono scaricati durante l'inizializzazione del consuntivazioni.component chiamando il metodo *getProgetti* del consuntivazioni.service. Nel web service questa richiesta viene gestita dall'action method *GetProgettiInCons* del ProgettoQueryController. Se l'utente ha il ruolo Responsabile o Risorsa team per l'accesso all'applicazione, verranno scaricati solo i progetti in cui esso è presente come Progetto-Risorsa o che appartengono a commesse di cui è responsabile commessa. Questo controllo è effettuato all'interno di IProgettoService tramite il service IRolesService. Se l'utente è un amministratore allora la richiesta è soddisfatta dal GetProgettiForDropdownForAdminInConsQueryHandler, mentre se è un responsabile o una risorsa team viene utilizzato il GetProgettiForDropdownInConsQueryHandler. Selezionato un progetto tra quelli disponibili, il consuntivazioni.component chiama il metodo *GetConsuntivazioni* del consuntivazioni.service per popolare la griglia delle consuntivazioni. La "Figura 36" mostra un esempio del genere.

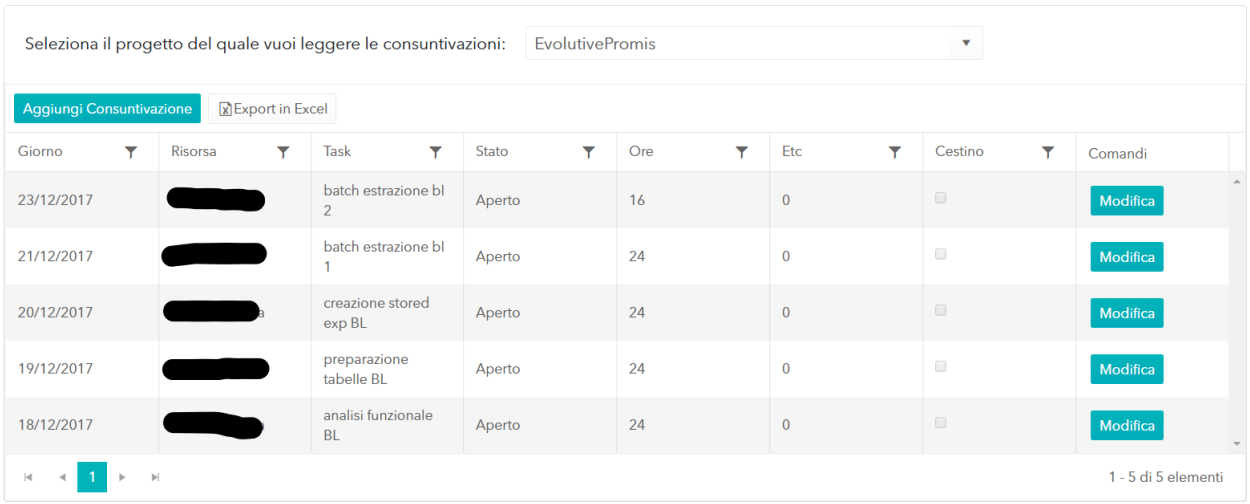


***Figura 36**: Griglia Consuntivazioni*

Cliccando su "Aggiungi Consuntivazione" o "Modifica" viene mostrato un pop-up, come mostrato nella "Figura 37". Se il bottone premuto è "Modifica" i campi del form sono riempiti con i dati della consuntivazione che si sta modificando. Per popolare la sorgente dati del campo Stato (Kendo Combobox) viene chiamato il metodo *getStati*, messo a disposizione dal consuntivazioni.service. Lato server, la richiesta è gestita dall'action method *GetStatiConsuntiviForDropdown* del controller StatiConsuntiviQueryController. Questo invoca IStatoConsuntivoService che delega la lettura degli stati al GetStatiConsuntiviForDropdownQueryHandler. Contemporaneamente il metodo *getRisorse* del consuntivazioni.service permette di popolare il campo Risorsa, che è un componente Kendo Combobox, scaricando le risorse già assegnate al progetto (modellate da modelRisorsa.ts). Questa richiesta è gestita dall'action method *GetRisorseWithMatricolaByProgetto* del controller RisorseQueryController. La richiesta viene passata al service IRisorsaService che ottiene le informazioni dal database tramite il GetRisorseWithMatricolaByProgQueryHandler. Selezionata una risorsa, viene chiamato il metodo *getTask*, fornito sempre dal consuntivazione.service, in modo da scaricare i task (modellati da modelTask.ts) del progetto assegnati alla risorsa scelta. In questo modo viene popolata la sorgente dati del campo task, che è un componente Kendo Combobox. Tramite un Kendo Switch l'utente può decidere se mostrare nella Combobox solo i task non completati (cioè con Etc maggiore di zero), o tutti i task, includendo anche quelli completati. Alla pressione del pulsante "Salva" viene chiamato il metodo *addConsuntivazione*, nel caso di inserimento, oppure *updateConsuntivazione*: entrambi sono metodi esposti dal consuntivazini.service. Lato server, sia in caso di aggiunta che di modifica, viene chiamato in causa IProgettoRepository perché si sta agendo su entità che fanno parte dell'Aggregate Root Progetto. Quindi, il repository costruisce e popola il progetto (oggetto del dominio) a cui fa riferimento la consuntivazione e invoca il metodo della classe Progetto specifico per l'operazione richiesta. Se nessuna eccezione viene sollevata il dato viene reso persistente tramite l'utilizzo dell'Entity Framework.

Nel caso di modifica, se all'utente è associato un Progetto-Risorsa con profilo Risorsa Team all'interno del progetto, l'operazione viene consentita solo se l'utente è l'autore della consuntivazione che si sta tentando di modificare, a prescindere dal fatto che il suo ruolo di accesso per l'applicazione possa essere anche quello di Responsabile. Questo controllo viene

effettuato all'interno di IConsuntivazioneService prima di invocare il repository IProgettoRepository.

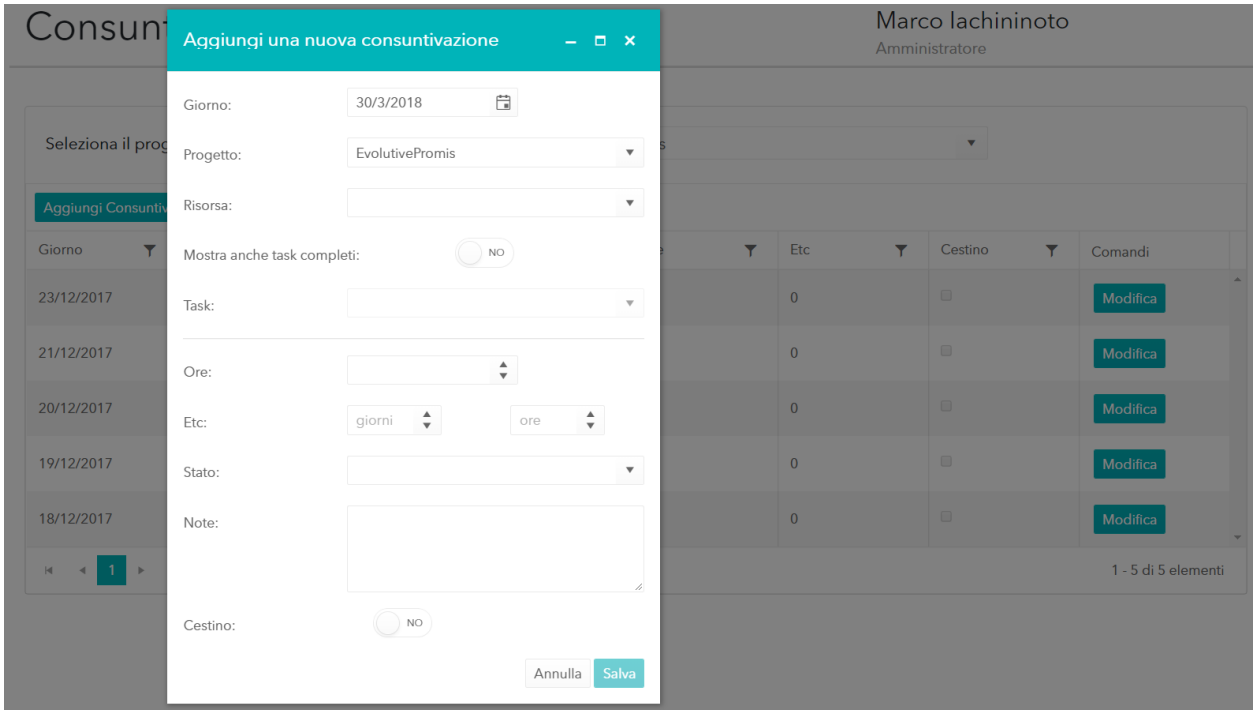


***Figura 37**: Inserimento consuntivazione*

5.7 Visualizzazione messaggi di errore

Il modulo SharedModule contiene i componenti responsabili della visualizzazione di eventuali errori ricevuti dal web service. All'interno di questo modulo sono stati definiti un service, alert.service, ed un component, notification.component, che permettono di gestire questo meccanismo di notifiche. Ogni risposta HTTP viene processata dall'http-interceptor, che verifica se la risposta contiene dei messaggi di errore. In questi casi, viene invocato l'alert.service per mostrare a video il messaggio di errore trasportato dalla risposta HTTP.

Innanzitutto, il costruttore del service effettua un subscribe all'evento associato ad ogni cambiamento di route, così che gli eventuali messaggi già mostrati vengano rimossi automaticamente ad ogni cambiamento di pagina.

Il service espone un metodo, *getAlert,* che ritorna un Observable; il notification.component durante la sua inizializzazione chiama questo metodo ed effettua un'operazione di subscribe nei confronti dell'oggetto Observable ritornato dal service. In questo modo il notification.component sarà avvisato ogni qual volta l'http-interceptor ordina la visualizzazione di un nuovo messaggio di errore chiamando il metodo *warn* esposto dall'alert.service. Questo metodo infatti aggiunge un nuovo messaggio per l'Observable a cui si era sottoscritto il notification.component.

6. Conclusioni

A conclusione di questo elaborato, in cui è stata realizzata una web application per la gestione dei progetti chiavi in mano, è possibile evidenziare i punti di forza della soluzione descritta e gli eventuali sviluppi futuri.

Seguire l'approccio tipico del Domain-driven design ha permesso la realizzazione di un'applicazione modulare e facilmente estensibile. Nel caso in cui sviluppi futuri necessitino di aggiungere delle nuove funzionalità o di modificare quelle già esistenti, il punto principale in cui dover agire sarà l'insieme delle classi di dominio. Il cuore della business logic dell'applicazione è infatti rappresentato dalle proprietà e dai metodi esposti dalle classi di dominio. Per questo non è stato necessario scrivere alcuna stored procedure, divincolando l'applicazione sviluppata dal livello di infrastruttura sottostante e quindi dalla tecnologia utilizzata per rendere persistenti i dati. Tutto il codice che serve per regolare la lettura dei dati dal database o l'accesso in modifica è distribuito tra le classi di dominio, che implementano le business rules, e i Query Handler o Repository, che utilizzano l'Entity Framework per l'accesso ai dati. In questo modo si evita al programmatore di dover scrivere query specifiche dipendenti dal tipo di database utilizzato.

Così come non esiste alcuna dipendenza dal database utilizzato, non esiste alcuna dipendenza da come è stato implementato il front-end. Il fatto che il server metta a disposizione i dati tramite chiamate alle API da esso esposte e che queste chiamate ritornino dati JSON, lascia massima libertà sulla scelta della tecnologia da utilizzare per lo sviluppo client-side e su come organizzare l'interfaccia utente. Questo vuol dire che le stesse funzionalità potrebbero essere offerte anche da un'applicazione mobile, sia essa Android o IOS, che usa le stesse API chiamate dall'applicazione Angular sviluppata nel corso di questa tesi. L'utilizzo di un framework recente e molto supportato come Angular, che ha dietro di sé il supporto ufficiale di Google e una community di developer molto ampia, garantisce che l'applicazione realizzata per il front-end non diventi obsoleta in tempi troppo brevi. L'ampio utilizzo della libreria Kendo UI for Angular ha inoltre permesso di mostrare i dati in formato tabellare sfruttando i potenti widget da essa offerti, dando all'utilizzatore dell'applicazione la possibilità di filtrare e ordinare i dati visualizzati come meglio crede.

Per quanto riguarda possibili sviluppi futuri si potrebbe arricchire la generazione di report sull'avanzamento dei progetti in modo da tenere conto non solo dell'aspetto temporale, ma anche di quello economico. Inoltre, si potrebbe aggiungere una funzionalità che permetta di gestire in modo semplice e intuitivo il piano ferie delle risorse che lavorano ai vari progetti. Infine, si potrebbe pensare anche alla possibilità di integrare il caricamento del timesheet aziendale con il contenuto della tabella relativa alle consuntivazioni.

Lo svolgimento di questa tesi mi ha permesso di trascorrere cinque mesi in azienda tramite i quali mi sono avvicinato al mondo lavorativo con cui mi dovrò confrontare appieno a seguito della laurea. Dato che lo sviluppo dell'applicazione è stato interamente affidato a me non ho potuto accumulare molta esperienza per quel che riguarda il vero e proprio lavoro di team. Comunque, ho imparato molto su come si vive e si lavora all'interno di una realtà aziendale

osservando tutto ciò che accadeva intorno a me. È stata quindi un'esperienza che rifarei volentieri e che consiglio a tutti coloro che si trovano nelle fasi finali dei propri studi e che non hanno mai avuto modo di confrontarsi con una realtà lavorativa.

Tramite lo sviluppo dell'applicazione analizzata nel capitolo precedente ho avuto modo di apprendere da zero e in modo abbastanza approfondito nuove tecnologie che non avevo mai utilizzato, come Angular. Contemporaneamente, mi sono confrontato con tecnologie che avevo già incontrato durante gli studi universitari, come la piattaforma .NET e il linguaggio C#, ma con le quali non avevo la stessa dimestichezza che ritengo di aver acquisito nel corso di questa esperienza. Inoltre, precedentemente non mi ero mai trovato di fronte alla necessità di utilizzare un framework per il mapping object-relational: solo con lo sviluppo di questa applicazione ho potuto apprendere i vantaggi che scaturiscono dall'utilizzo di un framework del genere, vantaggi che sono stati descritti nel corso del quarto capitolo e che non riguardano solamente l'applicazione in sé, ma anche l'attività di sviluppo del developer.

Ringraziamenti

Innanzitutto, ringrazio il mio Relatore, il Chiar.mo Professor Giorgio Bruno, e il mio tutor aziendale, Andrea Ricossa, per la cortese disponibilità mostratami in questo percorso di tesi. Un ringraziamento particolare va poi alla mia famiglia, cui dedico il lavoro svolto, e a tutti coloro che mi sono stati vicini durante il mio percorso accademico.

Sitografia

[1] https://it.wikipedia.org/wiki/Applicazione_web

[2] https://en.wikipedia.org/wiki/Web_application

[3] http://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app

[4] https://www.maxcdn.com/one/visual-glossary/web-application/

[5] https://en.wikipedia.org/wiki/Ajax_(programming)

[6] https://en.wikipedia.org/wiki/HTML5#New_APIs

[7] https://it.wikipedia.org/wiki/Architettura_multi-tier

[8] https://it.wikipedia.org/wiki/Model-View-Controller

[9] http://www.claudiodesio.com/ooa&d/mvc.htm

[10] http://www.html.it/pag/18299/il-pattern-mvc/

[11] https://www.nephila.it/it/blog/2014/10/03/web-framework-definizione/

[12] https://it.wikipedia.org/wiki/Framework_per_applicazioni_web

[13] https://en.wikipedia.org/wiki/ASP.NET#Extensions

[14] https://it.wikipedia.org/wiki/ASP.NET

[15] https://en.wikipedia.org/wiki/.NET_Framework

[16] http://www.tutorialsteacher.com/linq/why-linq

[17] http://www.tutorialsteacher.com/linq/what-is-linq

[18] http://www.tutorialsteacher.com/linq/linq-api

[19] http://www.tutorialsteacher.com/linq/linq-query-syntax

[20] http://www.tutorialsteacher.com/linq/linq-method-syntax

[21] https://it.wikipedia.org/wiki/Piattaforma_Java

[22] https://en.wikipedia.org/wiki/Java_(software_platform)#Web_server_and_enterprise_use

[23] https://en.wikipedia.org/wiki/Java_virtual_machine

[24] https://it.wikipedia.org/wiki/Java_EE

[25] https://www.seguetech.com/net-vs-java/

[26] https://www.rishabhsoft.com/blog/dot-net-vs-java

[27] https://it.wikipedia.org/wiki/Angular

[28] https://angular.io/guide/architecture

[29] https://it.wikipedia.org/wiki/TypeScript

[30] https://en.wikipedia.org/wiki/TypeScript

[31] http://www.html.it/pag/55620/introduzione-a-typescript/

[32] http://www.tutorialsteacher.com/webapi/what-is-web-api
[33] http://www.dotnettricks.com/learn/webapi/what-is-web-api-and-why-to-use-it-
[34] https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/
[35] http://www.tutorialsteacher.com/webapi/web-api-controller
[36] http://www.tutorialsteacher.com/webapi/web-api-routing
[37] http://www.tutorialsteacher.com/webapi/parameter-binding-in-web-api
[38] http://www.tutorialsteacher.com/webapi/action-method-return-type-in-web-api
[39] http://www.tutorialsteacher.com/webapi/request-response-data-formats-in-web-api
[40] http://www.tutorialsteacher.com/webapi/web-api-filters
[41] https://en.wikipedia.org/wiki/Microsoft_SQL_Server
[42] https://it.wikipedia.org/wiki/Transact-SQL
[43] http://www.entityframeworktutorial.net/what-is-entityframework.aspx
[44] http://www.entityframeworktutorial.net/basics/how-entity-framework-works.aspx
[45] http://www.entityframeworktutorial.net/EntityFramework-Architecture.aspx
[46] http://www.entityframeworktutorial.net/basics/context-class-in-entity-framework.aspx
[47] http://www.entityframeworktutorial.net/Types-of-Entities.aspx
[48] http://www.entityframeworktutorial.net/lazyloading-in-entity-framework.aspx
[49] http://www.entityframeworktutorial.net/eager-loading-in-entity-framework.aspx
[50] http://www.entityframeworktutorial.net/basics/entity-states.aspx
[51] http://www.entityframeworktutorial.net/choosing-development-approach-with-entity-framework.aspx
[52] https://en.wikipedia.org/wiki/Best_coding_practices#Testing
[53] https://it.wikipedia.org/wiki/SOLID
[54] https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
[55] https://it.wikipedia.org/wiki/Principio_di_singola_responsabilit%C3%A0
[56] https://en.wikipedia.org/wiki/Single_responsibility_principle
[57] https://it.wikipedia.org/wiki/Principio_aperto/chiuso
[58] https://en.wikipedia.org/wiki/Interface_segregation_principle
[59] https://it.wikipedia.org/wiki/Principio_di_inversione_delle_dipendenze
[60] https://it.wikipedia.org/wiki/Qualit%C3%A0_del_software#Manutenibilit%C3%A0
[61] https://it.wikipedia.org/wiki/Usabilit%C3%A0_del_web
[62] https://blog.angular-university.io/why-angular-angular-vs-jquery-a-beginner-friendly-explanation-on-the-advantages-of-angular-and-mvc/

[63] https://blog.codewithdan.com/2017/08/26/5-key-benefits-of-angular-and-typescript/
[64] https://angular.io/guide/aot-compiler#why-compile-with-aot
[65] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/
[66] https://turbonomic.com/blog/microsoft-sql-server-advantages-challenges-virtualization-admins-view/
[67] http://www.entityframeworktutorial.net/code-first/migration-in-code-first.aspx
[68] https://en.wikipedia.org/wiki/Single-page_application
[69] https://it.wikipedia.org/wiki/Domain-driven_design
[70] http://www.mokabyte.it/2008/11/domaindriven-1/
[71] http://www.mokabyte.it/2008/12/domaindriven-2/

Bibliografia

[1] Pro ASP.NET MVC 4, Freeman, Adam, Sanderson, Steven
[2] Domain-Driven Design: Tackling Complexity in the Heart of Software, Eric Evans